大数据搜索与挖掘及可视化管理方案

——**Elastic Stack 6**：Elasticsearch、Logstash、Kibana、X-Pack、Beats

高凯 岳重阳 江跃华 编著

(第4版)

清华大学出版社

大数据搜索与挖掘及可视化管理方案

——Elastic Stack 6：Elasticsearch、Logstash、Kibana、X-Pack、Beats（第 4 版）

高　凯　岳重阳　江跃华　编著

清华大学出版社
北　京

内 容 简 介

本书系统地介绍数据搜索与实时分析引擎套件 Elastic Stack 的相关技术，并通过实战讲解的方式介绍 Elasticsearch、Logstash、Kibana、X-Pack、Beats 等的应用。全书共 11 章，内容涵盖 Elasticsearch 架构简介、文档索引及管理、信息检索与聚合、Elasticsearch API 及其应用、Elasticsearch 配置与集群管理、基于 Logstash 的日志处理、基于 Kibana 的数据分析及可视化、基于 X-Pack 的系统运行监控、基于 Beats 的数据解析传输，最后给出两个信息检索与分析实例。

本书强调实践和面向初学者，力求反映基于 Elastic Stack 6 架构的最新成果。本书可供高等学校计算机科学与技术、软件工程、物联网、信息管理与信息系统等专业的学生在学习和科研中参考。对于从事大数据搜索与挖掘、日志分析、信息可视化、集群管理与性能监控的工程技术人员和希望了解网络信息检索技术的人员也具有较高的参考价值和工程应用价值。

图书在版编目(CIP)数据

大数据搜索与挖掘及可视化管理方案：Elastic Stack 6：Elasticsearch、Logstash、Kibana、X-Pack、Beats/高凯，岳重阳，江跃华编著. —4 版. —北京：清华大学出版社，2019
ISBN 978-7-302-50799-4

Ⅰ. ①大…　Ⅱ. ①高… ②岳… ③江…　Ⅲ. ①信息检索—研究 ②数据采集—研究　Ⅳ. ①G254.9 ②TP274

中国版本图书馆 CIP 数据核字(2019)第 061058 号

责任编辑：焦　虹　战晓雷
封面设计：傅瑞学
责任校对：时翠兰
责任印制：丛怀宇

出版发行：清华大学出版社
网　　址：http://www.tup.com.cn，http://www.wqbook.com
地　　址：北京清华大学学研大厦 A 座　　**邮　　编**：100084
社 总 机：010-62770175　　**邮　　购**：010-62786544
投稿与读者服务：010-62776969，c-service@tup.tsinghua.edu.cn
质量反馈：010-62772015，zhiliang@tup.tsinghua.edu.cn
课件下载：http://www.tup.com.cn，010-62795954
印 装 者：北京嘉实印刷有限公司
经　　销：全国新华书店
开　　本：185mm×240mm　　**印　　张**：24.25　　**字　　数**：497 千字
版　　次：2015 年 6 月第 1 版　　2019 年 8 月第 4 版　　**印　　次**：2019 年 8 月第1 次印刷
定　　价：59.00 元

产品编号：080619-01

Foreword

第4版 前 言

伴随着海量数据管理技术在国民经济以及互联网＋、物联网、移动计算等各个领域的广泛应用，分布式大数据搜索、日志分析与挖掘、数据可视化、集群管理与性能监控等问题正日益受到 IT 人员的普遍关注。开源的、基于 Lucene 的全文搜索引擎 Elasticsearch 以其独到的分布式数据处理能力发挥了重要作用。根据国际权威数据库产品评测机构 DB-Engines 统计，从 2016 年 1 月起，Elasticsearch 已超过 Solr 等成为排名第一的搜索引擎类应用，并且增长势头目前仍非常强劲。在 Elasticsearch 基础上，还衍生出 Logstash、Kibana、Beats、X-Pack、Elastic Cloud、Alerting、Graph、Reporting、ES-Hadoop等诸多相关组件，它们构成了 Elastic Stack 工具集的核心，为编程人员提供了分布式、可扩展的信息存储和全文检索机制、基于 Logstash 的日志处理机制、基于 Kibana 的挖掘结果可视化机制等。不仅如此，目前还出现了 Security（安全和管理插件，如权限控制、加密通信、审计等）、Monitoring（性能监控平台等）、Beats（提供了在应用服务器间传输事务信息的 Packetbeat、从服务器端传送日志的 Filebeat、分时段采集服务器上操作系统和服务的各项指标的 Metricbeat、负责传输 Windows 事件日志的 Winlogbeat）等中间件。在实时大数据处理的应用中，上述软件通常配合使用。Elastic Stack 可帮助企业搜索庞大的非结构化信息数据库。目前，Elasticsearch 拥有几千个重要客户，覆盖 80 多个国家，从一线互联网公司到传统行业，都能看到 Elasticsearch 的身影，全球很多著名公司（如 Facebook、eBay、IBM、Microsoft、Uber、Cisco、GitHub、携程网等）都在使用它。Elastic 公司的开源技术越来越受到众多开发者的青睐，已成为大数据领域分析工具的最佳选择。现在用户可以在谷歌 GCP 云平台上运行 Elastic Cloud 产品，执行搜索、分析以及可视化任务。从实战的角度掌握 Elasticsearch、Logstash、Kibana、X-Pack、Beats 等的基本使用方法和入门技巧很有必要。

从 2015 年开始，我们开始介绍和普及 Elastic Stack 的相关技术，并陆续出版了《实战 Elasticsearch、Logstash、Kibana——分布式大数据搜索与日志挖掘及可视化解决方案》（第 1 版）、《大数据搜索与日志挖掘及可视化方案——ELK Stack：Elasticsearch、Logstash、Kibana》（第 2 版）、《大数据搜索与日志挖掘及可视化管理方案——Elastic Stack 5：Elasticsearch、Logstash、Kibana》（第 3 版）、译著《精通 Elastic Stack》等多部技术书籍。短短几年，

Elastic Stack 发展很快，我们的上述图书也已重印了多次。我们参考了部分读者对本书第1版、第2版、第3版提出的意见，对其中的部分内容进行了必要的补充和修改、完善，对新推出的 Elastic Stack 6 进行了介绍。

同本书的第1版、第2版、第3版一样，第4版仍强调实践和面向初学者，并通过实战讲解的方式让读者更好地了解 Elasticsearch、Logstash、Kibana、X-Pack、Beats 等的应用。全书涵盖 Elasticsearch 的架构简介、文档索引及管理、信息检索与聚合、Elasticsearch API 及其应用（适当兼顾 J2EE 和 Python 用户）、Elasticsearch 配置与集群管理、基于 Logstash 的网络日志处理、基于 Kibana 的分析结果可视化、基于 X-Pack 插件的系统运行监控应用、基于 Beats 的数据解析传输、应用实例等。本书介绍的基于 Elastic Stack 6 架构的分布式大数据搜索、日志挖掘、可视化、集群管理与性能监控是入门方案，但对有一定基础的中、高级使用者也有一定的参考和工程应用价值。

本书第4版由高凯拟定写作大纲，高凯、岳重阳、江跃华合作完成，并由高凯审校了全书。在本书的写作过程中，编者得到了多方面的支持与帮助。高成亮、毛雨欣、聂颖杰、韩佳、谢宇翔、杨聪聪、侯雪飞、李明奇、杨铠成、朱玉、杨凯、李娇娥、徐倩、吴林芳等均给予了协助。编者在写作过程中参考了 Elastic 官方网站 https://www.elastic.co/以及互联网上众多热心网友提供的素材。本书的顺利完成也得益于参考文献中包含的相关工作及研究成果，在此谨向这些文献的作者、热心网友以及为本书提供帮助的人士致以诚挚的感谢。在本书写作过程中，编者也得到了清华大学出版社焦虹编审的大力支持和帮助，在此一并表示衷心感谢。

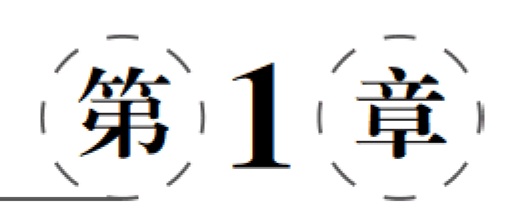
由于我们的学识水平有限，书中不妥之处在所难免，恳请广大读者批评指正。

编　者

2019年6月

Contents

目录



概　　述

> The open source Elastic Stack can reliably and securely take data from any source, in any format, and search, analyze, and visualize it in real time. (1) Elastic Cloud, provision and manage a fleet of Elastic Stack clusters (and X-Packs) on any infrastructure, all while monitoring and managing everything from a single pane of glass; (2) Built and maintained by Elastic engineers, X-Pack is a single extension that integrates handy features you can trust across the Elastic Stack; (3) Beats is a platform for lightweight shippers that send data from edge machines to Logstash and Elasticsearch; (4) Logstash is a dynamic data collection pipeline with an extensible plugin ecosystem and strong Elasticsearch synergy; (5) Elasticsearch is a distributed, JSON-based search and analytics engine designed for horizontal scalability, maximum reliability, and easy management; (6) Kibana gives shape to your data and is the extensible user interface for configuring and managing all aspects of the Elastic Stack; (7) APM finds performance bottlenecks in your applications; (8) ES-Hadoop can quickly query and get insight into your big data.
>
> https://www.elastic.co

随着大数据、大型电商网站以及 Web 2.0 技术的普及应用，越来越多的软件开发者需要进行海量信息的实时索引、检索，完成日志挖掘、可视化、性能监控等与信息检索、大数据搜索、挖掘相关的业务。虽然 Lucene 是许多互联网公司的标准信息检索工具之一，但它通常不提供实时检索，不具备良好的可扩展性，一般也不适合针对云计算环境的大数据搜索、挖掘及相应的数据管理工作。

Elastic Stack 是以 Elasticsearch、Logstash、Kibana、Beats 等为主，并涵盖 X-Pack、Elastic Cloud、Security(以前称为 Shield)、Alerting(通过 Watcher)、Monitoring(以前称为

Marvel）、Graph、Reporting、ES-Hadoop 等大数据处理与集群管理的工具集，也是目前流行的大数据分析的开源解决方案之一。2018 年 10 月，Elastic 公司在美国纽约证券交易所上市，这表明 Elastic 公司提供的大数据搜索和分析组件已受到行业的广泛认可，也给了数据搜索应用的开发者很大的鼓舞。

以 Elasticsearch、Logstash、Kibana、Beats、X-Pack 等几个开源软件为主的数据处理工具链为编程人员提供了分布式、可扩展的信息存储机制、基于 Lucene 7.x 及以上版本的信息检索机制、基于 Logstash 的日志处理机制、基于 Kibana 的挖掘结果可视化工具、基于 Beats 的性能监控架构以及对 Alerting 等封装后形成的 X-Pack 等。在一个典型的使用场景中，可以用 Logstash 处理日志等信息并充当“数据搬运工”的角色，用 Elasticsearch 作为后台数据的分布式存储平台和全文检索工具，用 Kibana 作为前端的可视化展示，用 Beats 作为采集系统监控数据的代理，用 X-Pack 完成安全、警告、监视、图形和报告等功能。

在基于数据存储和管理的 Elastic Stack 生态链中，往往也存储诸如产品信息、用户资料、文档、日志等可能涵盖对象（实体、人员、角色或者机器等）之间的引用关系的数据。Elastic Stack 为数据分布式存储、可视化查询、日志解析和系统性能监控等创建了一个功能强大的管道链，它们互相配合，共同完成大数据分析处理工作。很多国际知名企业都在使用 Elasticsearch 完成数据处理工作。例如，GitHub 已升级了其代码搜索程序，并将核心架构由 Solr 转向 Elasticsearch；Wikimedia 也启用了以 Elasticsearch 为基础的全新搜索框架。根据国际权威数据库产品评测机构 DB-Engines（http://db-engines.com/en）统计，Elasticsearch 已成为排名第一的搜索引擎类应用。学习 Elastic Stack，对于大数据处理、信息检索及搜索引擎研发、日志处理与分析、挖掘信息可视化、集群性能监控等具有重要现实意义。图 1.1 为截至 2019 年 1 月各搜索引擎类应用的排行榜。

18 systems in ranking, January 2019

Rank Jan 2019	Rank Dec 2018	Rank Jan 2018	DBMS	Database Model	Score Jan 2019	Score Dec 2018	Score Jan 2018
1.	1.	1.	Elasticsearch	Search engine	143.44	-1.26	+20.89
2.	2.	↑ 3.	Splunk	Search engine	81.43	-0.76	+17.42
3.	3.	↓ 2.	Solr	Search engine	61.48	+0.13	-2.89
4.	4.	4.	MarkLogic	Multi-model	14.26	-0.02	+3.05
5.	5.	5.	Sphinx	Search engine	7.69	-0.13	+1.44
6.	6.	6.	Microsoft Azure Search	Search engine	5.44	-0.24	+1.12
7.	7.	7.	Algolia	Search engine	3.49	-0.14	+0.54
8.	8.	8.	Google Search Appliance	Search engine	3.05	+0.12	+0.31
9.	9.	9.	Amazon CloudSearch	Search engine	2.49	-0.15	-0.20
10.	10.	10.	Xapian	Search engine	0.63	+0.03	+0.07
11.	11.	11.	CrateDB	Multi-model	0.50	+0.06	-0.01
12.	12.	12.	SearchBlox	Search engine	0.24	+0.00	+0.00
13.	13.	↑ 14.	searchxml	Multi-model	0.07	-0.01	+0.07
14.	14.	↓ 13.	DBSight	Search engine	0.05	-0.01	+0.04
15.	15.	↓ 14.	Manticore Search	Search engine	0.04	0.00	+0.04
16.	16.		FinchDB	Multi-model	0.03	+0.00	
17.	17.	↓ 14.	Exorbyte	Search engine	0.01	-0.01	+0.01
18.	18.	↓ 14.	Indica	Search engine	0.00	±0.00	±0.00

图 1.1　截至 2019 年 1 月各搜索引擎类应用的排行榜

1.1 Elasticsearch概述

Elasticsearch 是一个分布式的开源搜索与分析引擎，具有分布式集群的水平扩展、高可靠性、易于管理等诸多优点，能处理结构化、非结构化、时间序列等异构数据。Elasticsearch 来源于 Shay Bannon 的第一个开源项目 Compass。Compass 是一个基于事务的对象/搜索引擎映射和 Java 持久层框架，但 Elasticsearch 目前已经不再局限于单纯的搜索业务。作为开源分布式搜索与数据处理平台，Elasticsearch 不仅是一个库，还是一个基于 Lucene 构建的开源、分布、基于 RESTful 的信息检索框架，能够实时搜索，检索性能高，并采用 JSON 数据格式以及 Ruby DSL 设计模式，提供基于 Aggregations 的统计功能，提供便捷的部署和设置，集群可方便地扩展(可以扩展到上百台服务器，处理 PB 级别的结构化或非结构化数据，当然它也可以运行在单台 PC 上)；能对海量规模数据完成分布式索引与接近实时的信息检索，并提供多种管理工具，各种相关插件也可方便地集成到 Elasticsearch 中；它对外提供一系列基于 Java 和 HTTP 的 API，可用于分布式索引、检索、日志分析与数据挖掘等，且大多数配置是可修改的。Elasticsearch 语句可能包括如下几部分：

- 相应的 HTTP 请求方法或者变量(如 GET、POST、PUT、DELETE)。
- 集群中任意一个节点的访问协议、主机名以及端口。
- 请求的路径。
- 一个 JSON 编码的请求主体(如果需要)。

> Tips RESTful(REpresentational State Transfer)意即“表现层状态转化”，是目前流行的一种互联网软件架构，具有结构清晰、符合标准、易于理解、扩展方便等特点。这种架构下的每一个 URI 代表一种资源，客户端通过 GET(获取资源)、POST(新建或更新资源)、PUT(更新资源)、DELETE(删除资源)等方式来对服务器端的资源进行操作。在 Elasticsearch 中，RESTful 接口 URL 的格式是 http://<ip address>:9200/<index>/<type>/[<id>]。其中，index 可理解为数据库，type 可理解为数据表，id 相当于数据记录的主键。增、删、改分别对应 HTTP 请求的 GET、DELETE、PUT 方法。对 PUT 方法来说，调用时如果数据不存在，就创建它；如已存在，就更新它。Elasticsearch 可能会返回一个 HTTP 状态码(类似“200 OK”等)以及一个 JSON 格式的主体。

在使用 Elasticsearch 时，有很多基础服务可以用插件的方式来提供。这也是很多 Lucene 用户在面对大数据应用时转而使用 Elasticsearch 的原因之一，例如和 MongoDB、CouchDB 同步的 River 插件、中文分词插件、Hadoop 插件、脚本支持插件等。另外，

Elasticsearch 对分布式数据处理提供支持，其索引能拆分为多个分片，每个分片可有零个或多个副本，集群中的每个数据节点都可承载一个或多个数据分片，并且能协调和处理各种操作；负载再平衡（rebalancing）和路由选择（routing）在大多数情况下都是自动完成的。Elasticsearch 6. x 带来了许多增强功能和新特性，部分特点如下：

（1）使之能够从 Version 5 的最后一个版本滚动升级到 Version 6 的最后一个版本，不需要集群的完整重启，实现了无宕机在线升级、无缝滚动升级。

（2）Elasticsearch 6 能读取在 Elasticsearch 5.x 中创建的 Indices（但不能读取在 Elasticsearch 2. x 中创建的 Indices）。可以选择将数据保留在 Elasticsearch 5. x 群集中，并使用跨群集搜索同时在 Elasticsearch 6. x和 Elasticsearch 5. x 群集上进行搜索。

（3）Kibana 中的 X-Pack 插件提供了一个简单的用户界面，可帮助用户重新索引旧 Indices，以及将 Kibana、Security 和 Watcher 索引升级到 Version 6。它在现有集群上运行一系列检查，以帮助在升级之前更正问题。

（4）Elasticsearch 6. 0 以前，如果节点由于网络问题或节点重启而从集群断开连接，则节点上的每个分区都必须通过将分段文件与主分片进行比较并复制任何不同的分段来重新同步，耗时费力。Elasticsearch 6. 0 使用序列 ID，使得每个分片能重放该分片中缺少的操作，使恢复过程更加高效。

（5）通过索引排序，只要收集到足够的命中文档，搜索就可以终止，可以提高搜索效率。

Elasticsearch 的基本架构如图 1. 2 所示。可以看出，Elasticsearch 可以接收本地的、共享的和云平台上的数据。在 Lucene 提供的基本功能基础上，Elasticsearch 通过构建分布式索引完成对大数据的加工处理。其中，Zen 用于节点自动发现和 master 节点选举；EC2 (Elastic Compute Cloud，Elastic 计算云)借由提供 Web 服务的方式让使用者可弹性地运行自己的 Amazon 机器映像，提供可调整的云计算能力。用户可基于 RESTful 和客户端（如 Java 客户端）的方式，借助 Elasticsearch 提供的 API 完成数据操作、管理等工作。

1. 1. 1 Elasticsearch 的安装与简单配置

“工欲善其事，必先利其器。”要想了解 Elasticsearch，要从该软件的安装入手。传统上，Java 类的软件及使用往往比较烦琐，但 Elasticsearch 的安装却非常简单，几乎是“开箱即用”的。当然，前提是需要先下载 JDK 并配置相应的环境变量，同时确保系统可用内存大于 2GB。

下面对 Elasticsearch 的安装进行说明。进入官网 http://www. elastic. co，找到对应的 Elasticsearch 软件版本下载并解压。Elasticsearch 本身可以集成一些基本插件，在 GitHub 网站中有这些插件对应不同 Elasticsearch 版本的官方源码，可以按照网站上的简介来安装使用。

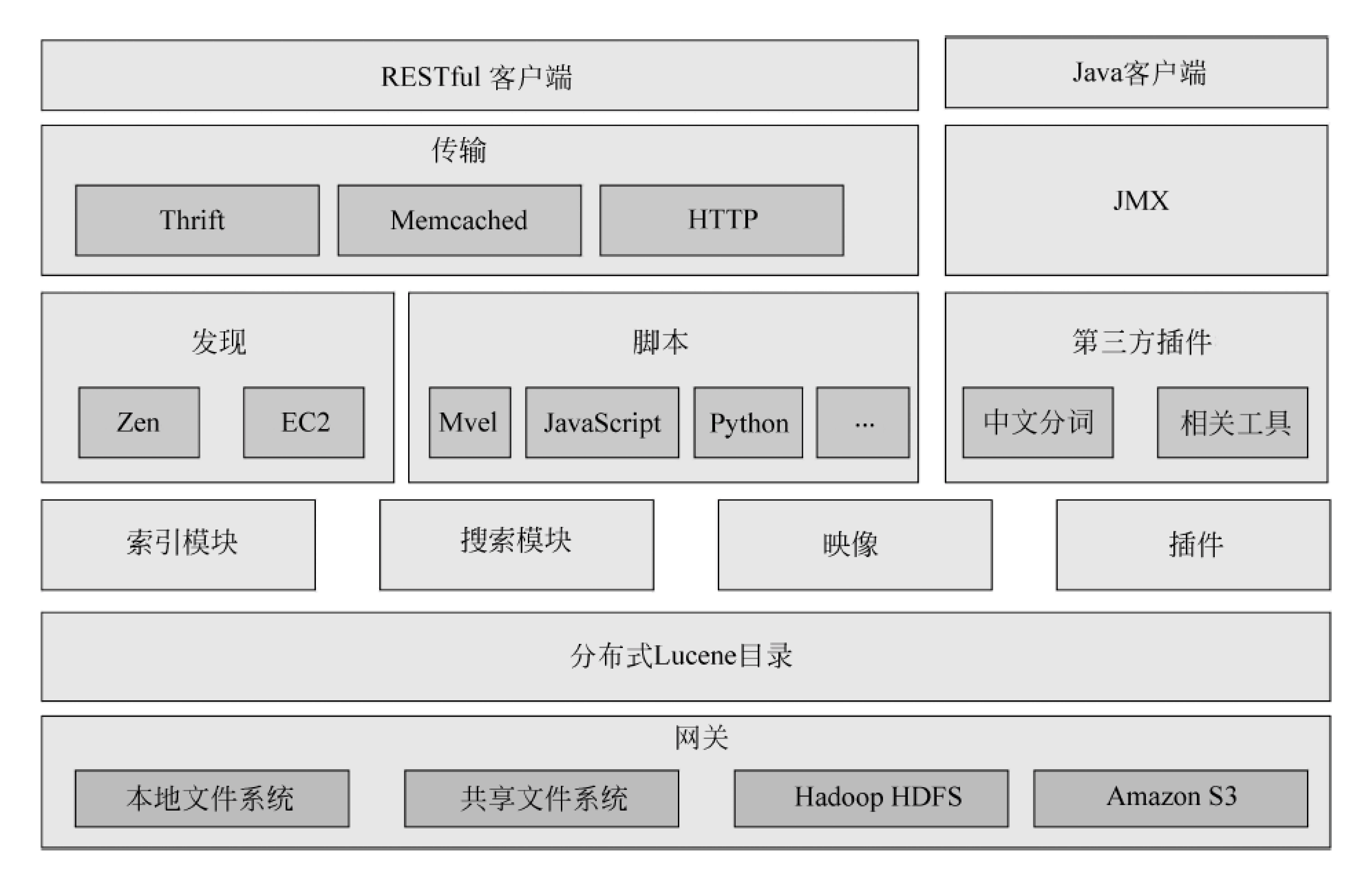


图 1.2 Elasticsearch 的基本架构

Elasticsearch 索引文件的大小只是原始文件大小的一部分，这可为集群节省服务器硬件采购费用。Elasticsearch 的 config 文件夹里面有 3 个配置文件：elasticsearch.yml 是基本配置文件，jvm.options 是 Java 虚拟机配置文件，log4j2.properties 是日志配置文件。下面简要介绍在 Ubuntu 系统上安装 Elasticsearch 6.2.3 的主要步骤。

首先，在官网上下载对应系统(如 Linux)的 Elasticsearch 软件包，进入安装文件所在目录，执行操作 tar -xvf elasticsearch-6.2.3.tar.gz 完成安装，具体步骤不再赘述。如果将 Elasticsearch 作为一个系统服务应用，可安装 Java Service Wrapper。该工具在其官网可以下载，网址是 http://wrapper.tanukisoftware.com/doc/english/download.jsp，限于篇幅，这里不再赘述。进入 Elasticsearch 的 bin 文件夹，执行./elasticsearch 命令，启动 Elasticsearch。

> Tips 若关闭 Elasticsearch，可在正在运行 Elasticsearch 的终端界面中按下组合键 Ctrl+C 来终止该节点的运行，此时该节点将会自动从群集中删除自身，将 translog 同步到磁盘，并执行其他相关的清理活动。如果 Elasticsearch 正作为一个系统服务应用运行，则应使用相应的系统服务管理程序来关闭。一个正确、有序的关闭操作可以确保 Elasticsearch 有机会清除和关闭未完成处理的资源。

之后，打开浏览器，输入类似 http://＜ip address＞：＜port＞，会显示类似图 1.3 的内容。其中：

（1）name：Elasticsearch 实例的名字，默认情况下是大小写字母和数字的组合，生成后长期留存，其设置同样是在 config/elasticsearch.yml 文件中完成的。

（2）version：版本号，以 JSON 格式表示的一组信息，其中的 number 字段代表当前运行的 Elasticsearch 版本号，build_snapshot 字段代表当前运行的版本是否是从源代码构建而来的，lucene_version 表示 Elasticsearch 基于的 Lucene 版本（图 1.3 显示该版本是基于 Lucene 7.2.1 而构建的）。

```
{
  "name" : "8FyHceb",
  "cluster_name" : "elasticsearch",
  "cluster_uuid" : "AGpcR3SxQ6uCT6tL1nxluw",
  "version" : {
    "number" : "6.2.3",
    "build_hash" : "c59ff00",
    "build_date" : "2018-03-13T10:06:29.741383Z",
    "build_snapshot" : false,
    "lucene_version" : "7.2.1",
    "minimum_wire_compatibility_version" : "5.6.0",
    "minimum_index_compatibility_version" : "5.0.0"
  },
  "tagline" : "You Know, for Search"
}
```

图 1.3　Elasticsearch 启动后的界面

（3）tagline：包含了 Elasticsearch 的第一个 tagline，内容为“You Know, for Search”。

图 1.3 中出现了 JSON 格式的数据。JSON（JavaScript Object Notation）是基于 JavaScript 的轻量级数据交换格式，是独立于语言的文本格式。在 JavaScript 中处理 JSON 数据不需要任何特殊的 API 或工具包，利用 JSON 可简单地表示半结构化数据，而且目前多数编程语言支持对 JSON 数据的解析。JSON 的基本语法表示如下：

（1）数据在用双引号表示的“"名称"："值"”对中，如“"name"："smith"”。

（2）可创建包含多个“"名称"："值"”的记录，如{"name"："smith"，"email"："abc@sjtu.org"}等，它表示以上两个数据是同一记录的一部分，数据间用逗号分隔，大括号中的内容表示对象，方括号中的内容表示数组。

以下代码是对相同数据的 XML 和 JSON 两种表示形式：

XML 示例代码：

```
<?xml version="1.0" encoding="utf-8"?>
<book>
    <name>Elasticsearch Searching
    </name>
    <author>
      <name>Gao</name>
      <sex>male</sex>
      <age>45</age>
      <country>China</country>
    </author>
    <price>10</peice>
</book>
```

JSON 示例代码：

```
{
  "book": {
    "name": " Elasticsearch Searching ",
    "author": {
      "name": "Gao",
      "sex": "male",
      "age": 45,
      "country": "China"
    },
    "price": 10,
  }
}
```

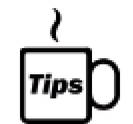
在 Elasticsearch 应用中，可以在很多地方看到 JSON 的身影。

1.1.2 Elasticsearch API 的使用方式

Elasticsearch API 有以下两种使用方式：

(1) 非客户端方式。通过 HTTP 方式的 JSON 格式进行调用。HTTP 的相关参数可在 elasticsearch.yml 中进行设置(出于安全考虑，也可禁用 HTTP 接口，只需在配置文件中将 http.enabled 设置为 false 即可)。

(2) 客户端方式。对 Java 来说，Elasticsearch 内置了传输客户端 TransportClient，它是一种轻量级传输客户端，可用来向远程集群发送请求。它并不加入集群本身，而是把请求转发给集群中的节点。客户端都使用 Elasticsearch 的传输协议，通过 9300 端口与 Java 客户端进行通信。集群中的各个节点也通过 9300 端口进行通信。



Elasticsearch 的 9200 端口是 HTTP 端口，9300 端口是传输端口。

1.2 Logstash

Logstash 是一款灵活、开源的数据转换和传递的管道，可处理多种信息来源的日志、事件、非结构化数据等，并可将处理后的数据输出到包括 Elasticsearch 等在内的多种数据存储目的地中。Logstash 本身并不产生日志，它仅是一个可接收多种多样的日志输入、经处理后转发到多个不同目的地的管道。Logstash 6.2 提供新的 API 插件，新增可单独配置的

密钥库，用于保护敏感信息。同时，6.2 版兼容性更好，适应面更宽。例如，基于 Logstash、Kafka（一个分布式的高可靠消息队列＋数据中转存储程序，可配置时间或大小来控制删除策略）、Elasticsearch、Kibana（可视化工具）的分布式信息处理架构，能有效管理分布式信息。Logstash 能够通过对更大体量和更多样的数据的利用来提升日志挖掘与分析能力。有关 Logstash 的使用方法，详见第 6 章。

1.3 Kibana

Kibana 是一款开源的数据可视化平台。在数据来源上，它能处理来源于 Elasticsearch、Logstash、Elasticsearch for Apache Hadoop and Spark、Beats 以及依赖于第三方技术的 Apache Flume、Fluentd 等在内的多种数据形式。Kibana 可以完成搜索、查看、与 Elasticsearch 分片中的数据进行交互等任务，也可轻松地在各种统计图、表格和地图中执行高级数据分析和数据可视化操作。Kibana 的出现使得对海量数据的理解成为可能，其简便、基于浏览器的界面能够让用户快速创建和共享动态仪表板。有关 Kibana 的使用方法，详见第 7 章。

1.4 Beats

可将 Beats 理解为一个代理(agent)，Beats 提供了用来实时收集日志的 Filebeat、用来收集系统基础设置数据的 Topbeat、用来统计收集网络信息的 Packetbeat 等，这些收集到的信息经过处理后可以输入到 Elasticsearch 中进行后续处理等（如图 1.4 中的左侧框图所示）。其中，Topbeat 用来收集系统负载、内存、硬盘以及每个进程的情况，它是跨平台的，会周期性发送指标到 Elasticsearch 中；Packetbeat 是开源的、分布式的组件，能实时嗅探每个事务的请求与响应并将相关数据插入 Elasticsearch，相关的社区也添加了对 MongoDB 的支持、基于 UDP 和 TCP 的 DNS 支持等。

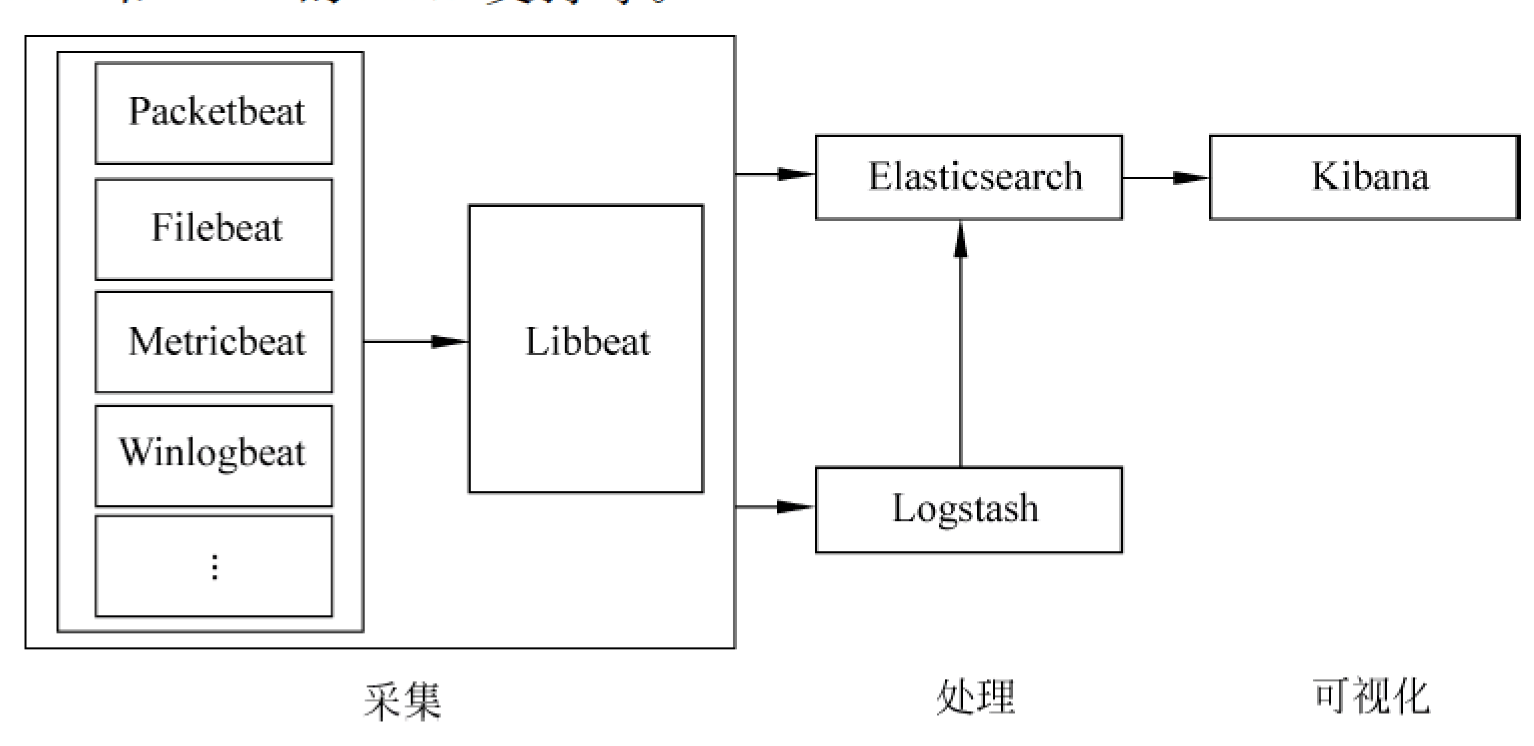


图 1.4　基于 Elastic Stack 的数据解决方案架构

1.5 X-Pack

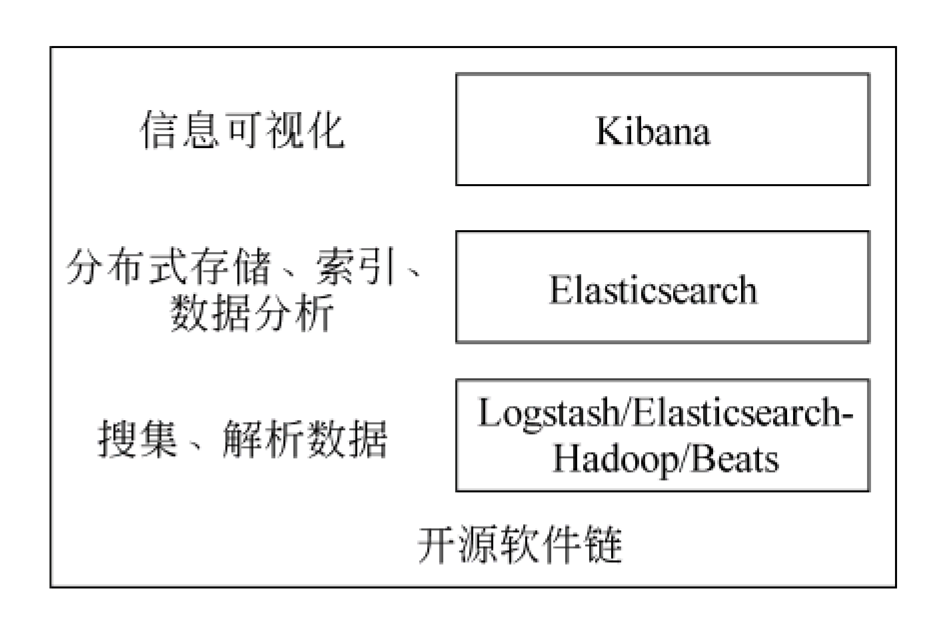
X-Pack 是 Elastic Stack 的扩展插件，提供安全、监控、告警、报告、图关联分析和机器学习六大扩展组件，无须额外安装插件，就能与 Elastisearch、Kibana 无缝对接。虽然 X-Pack 中的组件被设计为无缝协同工作，但仍可以轻松启用或禁用其中的部分功能。

1.6 其他

基于 Elastic Stack 的数据处理解决方案架构示意图如图 1.4 所示。

Elastic Stack 还包括其他一些组件。下面简要介绍最常用的两个组件。

(1) Elasticsearch for Apache Hadoop and Spark。虽然 Hadoop 生态链(如 Spark、Hive、Storm、MapReduce、Cascading 等)能提供多维度的大数据分析能力，但其快速、实时检索能力并不强大。Elasticsearch for Apache Hadoop and Spark 是位于存储大数据的 Hadoop 集群与 Elasticsearch 之间的中间件。例如，在图 1.5 所示的 Elastic Stack 相关软件产品结构中，它可以无缝连接 Hadoop 和 Elasticsearch 间的数据，能提供实时检索，通过 Kibana 更可方便地实现 Hadoop 生态链中的数据流可视化。



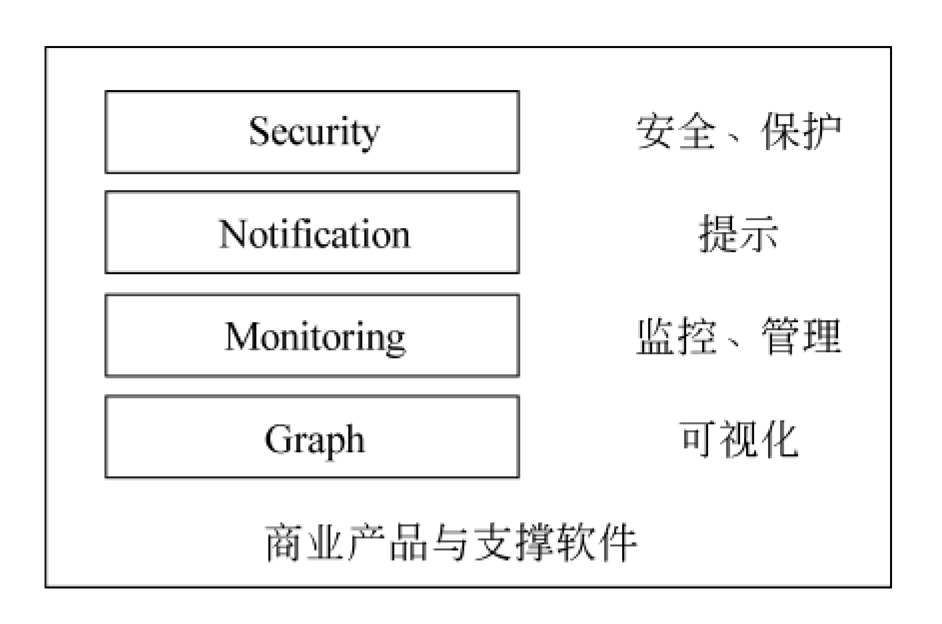


图 1.5 Elastic Stack 相关软件产品

(2) Elasticsearch Cloud。企业级管理平台，可配置、管理和监控任何规模、任何基础设施中的 Elasticsearch、Kibana 和 X-Pack，同时通过单个控制台管理一切。它通过可视化操作，将所有 Elastic 部署的规模调整、安全保护、升级和备份操作变得更加简单、高效。

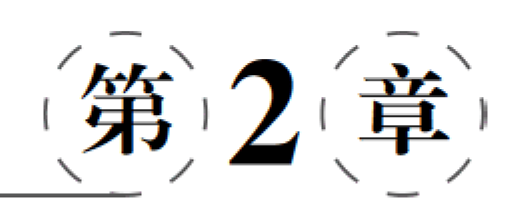
1.7 扩展知识与阅读

对大型网站来说，对于诸如“本周有多少新注册用户？”“广告的投放效果如何？”等统计工作来说，Elastic Stack 以其灵活的处理、简易的配置、近乎实时的检索、方便的集群可扩充

性、可视的数据挖掘结果等诸多优点，成为分布式搜索与日志挖掘及可视化的首选方案之一。从广义上说，Elasticsearch 也属于数据库范畴，但它能够轻松地进行大规模的横向扩展，以支持对 PB 级的结构化和非结构化海量数据的处理。由于关系型数据库对全文检索功能的支持相对不足（如传统关系型数据库并不能很好地解决大数据带来的问题，单机的统计和可视化工具也往往比较欠缺），所以一些实际项目需要将关系型数据库中的大数据同步到 Elasticsearch 中，以提供更加强大的全文检索功能。众所周知，Elasticsearch 和 Solr、Nutch 等都是基于 Lucene 构建的。了解 Lucene 的前世今生，更加有助于学习好 Elasticsearch。有关 Lucene 更详细的内容，参见文献（Michael，2011）。文献（韩陆，2014）介绍了基于 Java 标准规范实现 RESTful 的方法，这对于深刻理解 JAX-RS 标准和 API 设计，了解 Jersey 的使用要点和实现原理以及基于 RESTful 的 Web 服务的设计思想是很有帮助的。文献（杨传辉，2014）系统讲述了构建大规模存储系统的核心技术和原理，分析了 Google、Amazon、Microsoft 和阿里巴巴的大规模分布式存储系统的原理。有关现代信息检索发展、倒排索引构建、搜索引擎系统研发等可以参阅文献（Ricardo，2012）和文献（高凯，2014）。有关信息检索、搜索引擎等核心技术的通俗叙述，可以参阅文献（吴军，2013）。另外，Elasticsearch 扩展性非常好，有很多官方和第三方开发的插件，读者可以查阅 Elastic 官方技术文档、GitHub 开源库，获取最新信息。

1.8 本章小结

本章简要介绍了 Elastic Stack 的背景和组成。Elastic Stack 允许用户处理来源各异、种类不同的日志和数据，并以接近实时的可搜索的方式来处理信息和可视化分析结果。本章给出了背景知识，对 Elastic Stack 中出现的概念进行了说明。学习本章内容后，应该了解 JSON、RESTful 以及 Elasticsearch 插件等的用法。



文档索引及管理

> Elasticsearch is a highly scalable open-source full-text search and analytics engine. It allows you to store, search, and analyze big volumes of data quickly and in near real time. It is generally used as the underlying engine/technology that powers applications that have complex search features and requirements.
>
> https://www.elastic.co

在 Elasticsearch 中，对文档进行索引等操作时，既可以通过 RESTful 接口进行操作，也可以通过 Java 客户端进行操作。本章介绍基于 RESTful 的文档索引与管理方法，面向客户端的编程方法参见第 4 章。

21 文档索引概述

在信息检索过程中，文本数据首先要经过加工处理，才能被检索到，而这个加工处理过程就是建立索引文件(通常是倒排索引文件)。一般的信息检索过程是用户通过接口提交查询请求，在索引文件中检索出相关结果，并按相关度排序之后返回给用户。可见，建立文档索引是基础性的加工过程。由于索引文件通常要面向大量用户的查询，因此索引文件的设计要尽量高效，以便由索引项快速定位到相应文档。当前文档索引方法有倒排索引、后缀数组索引、签名文件索引等，其中倒排索引是用得最广的方法，在 Lucene 中采用的就是经典的倒排索引技术。目前成熟的信息检索系统几乎普遍采用这种索引方法。可以说，倒排索引是检索技术的基础。

倒排索引文件可看作是一种描述了一个词项集合 terms 中的元素和一个文档集合 docs 中的元素的对应关系的数据结构。若记 N 为文档集合的大小，M 为词项集合的大小，docs$=\{d_1,d_2,\cdots,d_N\}$，terms$=\{t_1,t_2,\cdots,t_M\}$，则倒排索引文件可给出 t_j 出现在哪些 d_i 中(或 t_j 在 d_i 中出现在哪些位置)的信息。一般地，倒排索引文件从逻辑上可分为两部分：一部分

用于表示文档的索引项；另一部分则由多个位置表组成，每个位置表和索引中的某个索引项相对应，并记录所有出现过该索引项的文档及该索引项在文档中的具体位置，以便计算出索引项之间的相邻关系。图 2.1 是一个倒排索引文件结构示意图，表示在文档 d_1 和 d_2 中出现过“应用”索引项，而在文档 d_2 和 d_3 中出现过“技术”索引项。由于 d_2 文档中上述两个索引项是相邻的（分别位于 51 和 52 位置），因此“应用技术”就应该出现在 d_2 文档中。如果要检索“应用技术”一词，则首先在倒排索引文件中找到包含“应用”一词的候选文档集合，然后从候选文档集合中筛选出在这个索引项后紧跟“技术”的文档（是 d_2，而非 d_1 或 d_3）。这种方式可以加快检索速度（高凯，2014）。

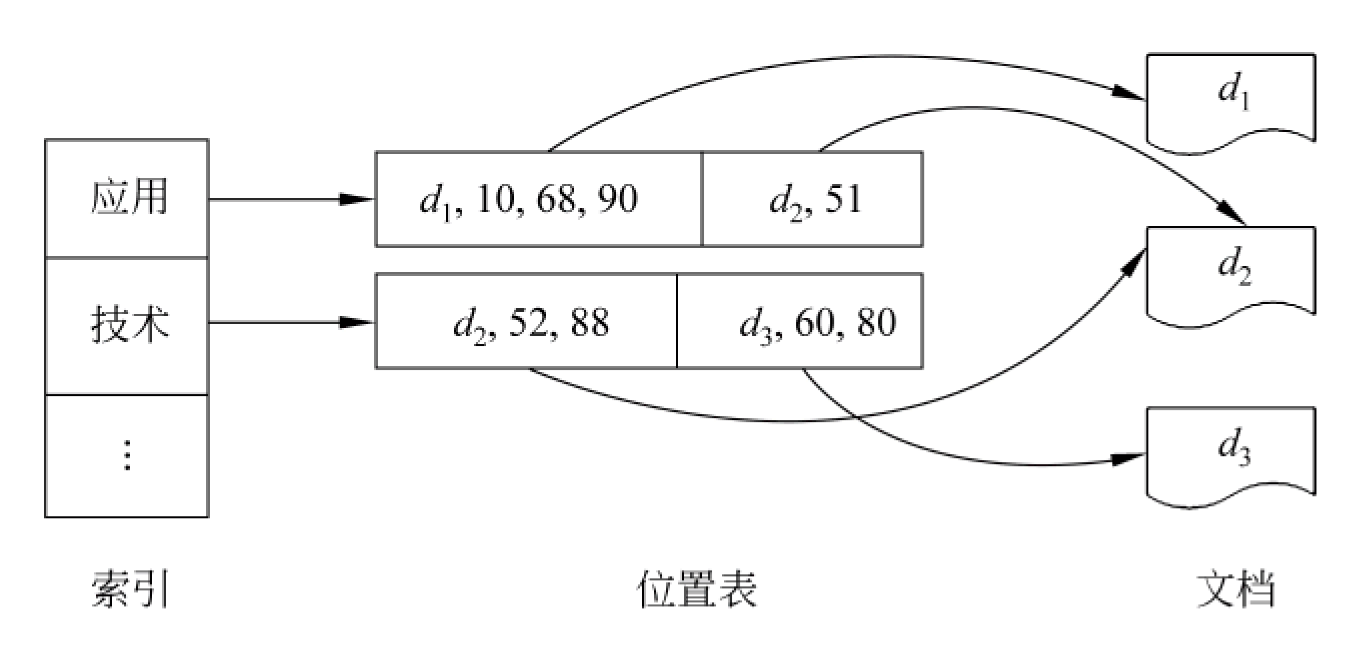


图 2.1　倒排索引文件示意图

Elasticsearch 实现准实时索引的关键就是将新收到的数据写到新的索引文件里。倒排索引机制会将文本信息切分成称为 token 的信息单元，再利用这些 token 构造倒排索引。索引文件一般由一个或多个子索引段（segment）组成，生成子索引段的数据则来源于内存中的缓存。除了子索引段外，索引文件中还有一个 commit 文件，用来记录索引内所有的子索引段信息。假设系统检测到当前索引有 M 个子索引段，对新接收的进入内存缓冲区中的数据，首先以默认值 1s 的时间间隔将其刷新到文件系统的缓存中去，这样系统即可准实时地检索到这个子索引段中的信息。Elasticsearch 也提供了单独的/_refresh 接口，如果需要缩短这个默认的 1s 时间间隔，可以通过调用这个接口来实现个性化的操作。同时，Elasticsearch 将数据定时输出到硬盘，子索引段数量增加到 $M+1$ 个，并同步更新 commit 文件。随着时间的推移，上述方法的弊端就会显现，即可能存在大量零散的子索引段文件。为此，Elasticsearch 会在后台运行相关的任务，定期地主动将这些零散的子索引段数据归并，尽量让索引只保留少量的、较大的子索引段文件。当归并完成后，检索请求将在这些归并后的较大的子索引段文件上进行。在具体实施中，归并线程是按照一定的策略来挑选子索引段进行归并的，相关技术细节可参阅文献（饶琛琳，2015）。

2.2 head：Elasticsearch用于数据管理的工具之一

head 是一种与 Elasticsearch 集群交互的 Web 前端工具。head 不是 Elastic 公司的系列化产品，但利用它，能够便捷地浏览和操作集群中的数据，为开发人员和用户提供直观的可视化界面。在 Elasticsearch 早期版本中，head 以插件的形式整合到其安装主目录下的 plugins 目录中；而在 Elasticsearch 5.0 及之后的版本中，head 变为独立运行的本地或服务器程序，用户可以在 GitHub 网站中下载源程序代码，在本地运行或搭建服务器启动 head。

> Tips 访问 GitHub 网站，在文件列表的左上方选择 master，将程序源代码 ZIP 压缩包保存到本地并解压。

如果让 head 作为服务器程序部署和运行，则需要借助 npm 和 Grunt 组件完成。npm 是随同 node.js 一起安装的包管理工具，能解决 node.js 代码部署上的很多问题。Grunt 是一个基于任务的 JavaScript 命令行构建工具，Grunt 及其插件是通过 npm 安装并管理的。

从互联网下载并安装 node.js（可使用查看软件版本的终端命令 node -v 和 npm -v 来测试其能否成功运行）。在确认 npm 工作正常之后，进入解压后的 elasticsearch-head-master 目录，使用终端命令 sudo npm install 安装该目录中的一些组件（包括 grunt 的一部分必要组件），部分安装过程如图 2.2 所示。

```
> phantomjs-prebuilt@2.1.16 install /home/cy/elasticsearch-head-master/node_modul
es/phantomjs-prebuilt
> node install.js

PhantomJS not found on PATH
Download already available at /tmp/phantomjs/phantomjs-2.1.1-linux-x86_64.tar.bz2
Verified checksum of previously downloaded file
Extracting tar contents (via spawned process)
Removing /home/cy/elasticsearch-head-master/node_modules/phantomjs-prebuilt/lib/p
hantom
Copying extracted folder /tmp/phantomjs/phantomjs-2.1.1-linux-x86_64.tar.bz2-extr
act-1529323566397/phantomjs-2.1.1-linux-x86_64 -> /home/cy/elasticsearch-head-mas
ter/node_modules/phantomjs-prebuilt/lib/phantom
Writing location.js file
Done. Phantomjs binary available at /home/cy/elasticsearch-head-master/node_modul
es/phantomjs-prebuilt/lib/phantom/bin/phantomjs
elasticsearch-head@0.0.0 /home/cy/elasticsearch-head-master
├─┬ grunt@1.0.1
│ ├── coffee-script@1.10.0
│ ├─┬ dateformat@1.0.12
│ │ ├── get-stdin@4.0.1
│ │ └─┬ meow@3.7.0
│ │   ├─┬ camelcase-keys@2.1.0
│ │   │ └── camelcase@2.1.1
```

图 2.2 使用 npm 安装服务器程序的组件

> Tips 这里对 head 的安装需要提升权限执行。Linux 用户需要在命令前加 sudo，并输入当前用户的密码；Windows 用户不必加 sudo，但需要以管理员身份运行控制台。

安装过程结束后，直接使用终端命令 npm run start 启动服务器，如图 2.3 所示。

```
cy@cy-N53SN:~/elasticsearch-head-master$ npm run start

> elasticsearch-head@0.0.0 start /home/cy/elasticsearch-head-master
> grunt server

(node:8673) ExperimentalWarning: The http2 module is an experimental API.
Running "connect:server" (connect) task
Waiting forever...
Started connect web server on http://localhost:9100
```

图 2.3　启动 head 服务器

从 Grunt 程序的输出信息中不难看出，该服务器占用 9100 端口，因此可在浏览器中输入 http://localhost:9100 来访问 head。在 Elasticsearch 启动时，head 将自动检测其来自 9200 端口的信息并自动完成连接 Elasticsearch 的工作。需要说明的是，为防范跨域脚本攻击，在尚未为 Elasticsearch 配置 cors 之前，head 不会与 Elasticsearch 连接。此时应在其安装主目录下的 config/elasticsearch.yml 配置文件末尾添加如下两行配置信息并重启 Elasticsearch：

```
http.cors.enabled: true
http.cors.allow-origin: "*"
```

> Tips　上述第二行配置信息中的 * 表示“允许从任何来源跨域访问”，这样的配置存在一定风险。如需要进行更安全的配置，可将 * 改为远程主机地址的正则表达式，以过滤不安全的访问来源。

如果仅需让 head 在本地运行，直接在浏览器打开 head 源程序目录中的 index.html 文件即可，但要注意做好上文提到的两行关于 cors 的配置。

完成上述过程之后，启动 Elasticsearch 并在 head 中检查连接是否正常。如果一切正常，那么 head 会显示和早期版本类似的界面，如图 2.4 所示，其中每一个小方格均代表一个数据分片（shard），横向每一行均代表一个节点（node），纵向每一列均代表一个索引（index）。

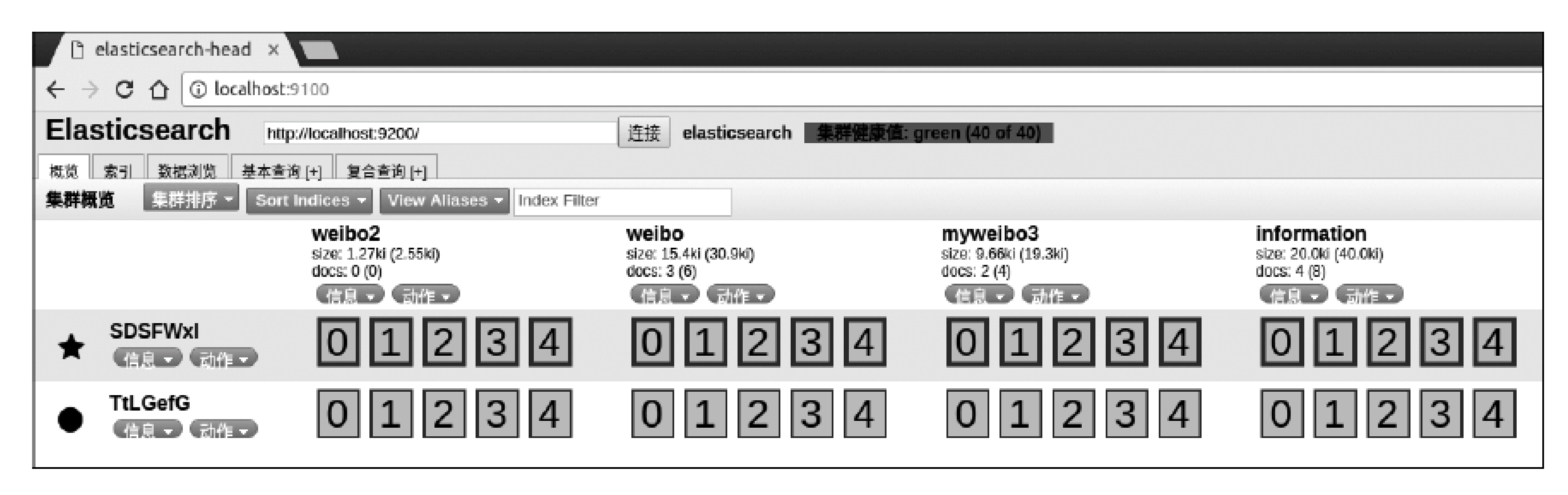


图 2.4　成功运行的 head

图 2.4 中开启了两个节点(包括一个主节点和一个从节点)。在初次安装 Elasticsearch 之后,若要开启两个或多个节点,应该预先在 elasticsearch.yml 配置文件末尾加入如下配置(表示允许最多开启 3 个节点)并重新开启各节点:

```
node.max_local_storage_nodes: 3
```

Tips head 并不是 Elastic 官方提供的可视化工具,为使 Elasticsearch 索引中的数据更加直观,查询更加简便,本书接下来的几章将继续使用 head 进行有关查询命令的介绍。Elastic 官方推出的 Kibana 及其 X-Pack 插件是与 Elasticsearch 配套的可视化管理方案,推荐读者在项目实战中使用带有 X-Pack 插件的 Kibana。

23 建立索引

在 Elasticsearch 中,可通过 index API 来对文档进行索引操作。在建立索引文件时,可以设置分布式索引文件的 shards 数量和 replicas(副本)数量。例如,可通过使用{index}/_settings(指定 index 名称的配置)子句修改索引文件的配置。

Tips 在执行下面的代码段 2.1 时,可以使用终端中的 curl 命令(见代码段 2.1,这里以 Ubuntu 系统为例,安装 curl 需使用终端命令 sudo apt install curl)。也可以使用 Elasticsearch 的 Web 前端,在该界面中提供了“复合查询”标签页,其中的“查询”窗口有 3 行输入框,在第一行填写 Elasticsearch 在 9200 端口的 URL 地址(如 localhost:9200),在第二行填写与索引相关的信息(可参照代码段 2.1),在第三行填写 JSON 格式的查询代码。此外,Elastic Stack 推出的 Kibana 也提供了专用于执行此类代码的 Dev Tools,其书写格式与 Elastic 官网一致。有关 Kibana 及 Dev Tools 中的代码书写格式的相关内容,将在第 7 章进行介绍。

代码段 2.1 实现了对名为 myweibo1 的索引文件的创建工作(注意是通过 PUT 方式向系统提交索引请求),方法中指定了索引数据的 shards 数量和 replicas 数量(如果不指定它们,系统会采用默认值),参数-d 后面是拟提交的索引数据。

代码段 2.1:使用 JSON 数据格式创建索引。执行后会新建一个名为 myweibo1 的新的索引文件

```
curl -H 'Content-Type: application/json' -XPUT 'http://localhost:9200/
myweibo1/' -d '{
```

```
    "settings" : {
      "index" : {
          "number_of_shards" : 5,
          "number_of_replicas" : 1
      }
    }
}'
```

Tips 在 Elasticsearch 6.x 及以后的版本中，向其发送的 curl 请求中必须显式加入 Content-Type 类型。本章给出的例子通过使用参数-H 'Content-Type:application/json' 指定了请求中传递的内容为 JSON 格式。

也可以采用_settings 子句实现其他相应功能。代码段 2.2 修改了 myweibo1 索引文件并将其 replicas 数量改为指定数值。图 2.5 显示该索引的副本已变成设定的数值了。

```
{
    "state": "open",
    "settings": {
        "index": {
            "creation_date": "1526659332943",
            "number_of_shards": "5",
            "number_of_replicas": "7",
            "uuid": "u8F91gZBRYOOtHgSK7P9gg",
            "version": {
                "created": "6020499"
            },
            "provided_name": "myweibo1"
        }
    },
```

图 2.5　索引文件属性信息

代码段 2.2：修改索引文件

```
curl -H 'Content-Type: application/json' -XPUT 'localhost:9200/myweibo1/
_settings' -d '{
  "index" : {
    "number_of_replicas" : 7
  }
}'
```

Tips 上述语句中的 number_of_replicas 参数用于设置当前索引的副本数量，它也可换成如下参数：

- blocks.read_only：如设为 true，则当前索引只允许读，不允许写或更新。
- blocks.read：如设为 true，则禁止读取操作。
- blocks.write：如设为 true，则禁止写操作。
- blocks.metadata：如设为 true，则禁止对 metadata 操作。

对于{index}/_settings 子句，如果选择的 HTTP 操作类型是 GET(见下面的 XGET 语句)，则可以获取当前索引文件的较为详细的配置信息。

```
curl -H 'Content-Type: application/json' -XGET 'http://localhost:9200/myweibo1/_settings'
```

返回值如下：

```
{
"myweibo1" : {
    "settings" : {
      "index" : {
        "creation_date" : "1526659332943",
        "number_of_shards" : "5",
        "number_of_replicas" : "7",
        "uuid" : "u8F91gZBRYOOtHgSK7P9gg",
        "version" : {
          "created" : "6020499"
        },
        "provided_name" : "myweibo1"
      }
    }
  }
}
```

采用如下语句，可一次性获得多个索引文件的配置信息(下面的例子返回 weibo、weibo2 这两个索引文件)：

```
curl -H 'Content-Type: application/json' -XGET 'http://localhost:9200/weibo, weibo2/_settings'
```

还可以使用_all 参数获取所有索引的配置信息：

```
curl -H 'Content-Type: application/json' -XGET 'http://localhost:9200/_all/
_settings'
```

也可以使用通配符获取一批索引的配置参数：

```
curl -H 'Content-Type: application/json' -XGET 'http://localhost:9200/weibo*/_settings'
```

也可以通过指定 JSON 格式的数据向指定的索引文件中插入数据并建立相应的索引。代码段 2.3 新建了一个索引文件(名为 myweibo3),并使用_doc 参数向其中写入 JSON 格式的字段信息(这些字段包括 user、post_date、mymessage 等)。此时在请求中未指定文档 id(Elasticsearch 将自动为这段信息分配一个随机 id)。注意该请求以 POST 方式提交,执行完毕后的结果如图 2.6 所示。

代码段 2.3: 向索引文件中插入 JSON 格式的数据,自动生成文档 id

```
curl -H 'Content-Type: application/json' -XPOST 'http://localhost:9200/
myweibo3/_doc' -d '{
  "user": "Alan",
  "post_date": "2018-01-01T08:00:00",
  "mymessage": "This is an example on creating an index"
}'
```

查询 20 个分片中用的 20 个. 1 命中. 耗时 0.004 秒

_index	_type	_id	_score ▲	user	post_date	mymessage
myweibo3	_doc	1m8gdGMB27N6d9FAnKf8	1	Alan	2018-01-01T08:00:00	This is an example on creating an index

图 2.6　向新建的索引文件中插入指定的信息

Tips　Elasticsearch 的内置字段主要有_all、_field_names、_id、_index、_meta、_parent、_routing、_source、_type、_uid 等,字段类型主要有 text/keyword、integer/long、float/double、boolean、date 等。在图 2.7 中显示了_index、_type、_id 等内置字段(其含义不再赘述)。图 2.7 中显示的其他字段(即 user、post_date、mymessage)是用户以 JSON 格式自定义的字段。

代码段 2.4 是向新建立的索引文件 myweibo3 中添加文档的语句,这个文档共有 3 个字段(分别是 user、post_date、mymessage)。和代码段 2.3 的语句不同的是,在 URL 中的参数 3 是为新建立的文档指定的 id 号,最后以参数_create 结束,注意此时该请求使用 PUT 方式提交。该语句的执行结果如图 2.7 所示。

代码段 2.4: 指定新插入索引数据的 id 号

```
curl -H 'Content-Type: application/json' -XPUT 'http://localhost:9200/
myweibo3/_doc/3/_create' -d '{
```

```
    "user": "LiMing",
    "post_date": "2018-01-01T14:00:00",
    "mymessage": "Hello Tom"
}'
```

查询 20 个分片中用的 20 个. 2 命中. 耗时 0.005 秒

_index	_type	_id	_score ▲	user	post_date	mymessage
myweibo3	_doc	1m8gdGMB27N6d9FAnKf8	1	Alan	2018-01-01T08:00:00	This is an example on creating an index
myweibo3	_doc	3	1	LiMing	2018-01-01T14:00:00	Hello Tom

图 2.7　向指定的索引文件中插入指定内容的信息

在建立好索引文件后，可以通过在终端中输入命令来获取指定索引文件的状态信息，例如 curl -H 'Content-Type:application/json' -XGET localhost:9200/{index}/_stats，注意这里使用了/_stats 参数。图 2.8 给出了查询索引文件返回的状态信息。


```
▼ "primaries": {
   ▼ "docs": {
       "count": 0,
       "deleted": 0
   },
   ▼ "store": {
       "size_in_bytes": 1150
   },
   ▼ "indexing": {
       "index_total": 0,
       "index_time_in_millis": 0,
       "index_current": 0,
       "index_failed": 0,
       "delete_total": 0,
       "delete_time_in_millis": 0,
       "delete_current": 0,
       "noop_update_total": 0,
       "is_throttled": false,
       "throttle_time_in_millis": 0
   },
   ▼ "get": {
       "total": 0,
       "time_in_millis": 0,
       "exists_total": 0,
       "exists_time_in_millis": 0,
       "missing_total": 0,
       "missing_time_in_millis": 0,
       "current": 0
   },
▼ "total": {
   ▼ "docs": {
       "count": 0,
       "deleted": 0
   },
   ▼ "store": {
       "size_in_bytes": 1150
   },
   ▼ "indexing": {
       "index_total": 0,
       "index_time_in_millis": 0,
       "index_current": 0,
       "index_failed": 0,
       "delete_total": 0,
       "delete_time_in_millis": 0,
       "delete_current": 0,
       "noop_update_total": 0,
       "is_throttled": false,
       "throttle_time_in_millis": 0
   },
   ▼ "get": {
       "total": 0,
       "time_in_millis": 0,
       "exists_total": 0,
       "exists_time_in_millis": 0,
       "missing_total": 0,
       "missing_time_in_millis": 0,
       "current": 0
   },
▼ "indices": {
   ▼ "myweibo1": {
      ▼ "primaries": {
         ▼ "docs": {
             "count": 0,
             "deleted": 0
         },
         ▼ "store": {
             "size_in_bytes": 1150
         },
         ▼ "indexing": {
             "index_total": 0,
             "index_time_in_millis": 0,
             "index_current": 0,
             "index_failed": 0,
             "delete_total": 0,
             "delete_time_in_millis": 0,
             "delete_current": 0,
             "noop_update_total": 0,
             "is_throttled": false,
             "throttle_time_in_millis": 0
         },
         ▼ "get": {
             "total": 0,
             "time_in_millis": 0,
             "exists_total": 0,
             "exists_time_in_millis": 0,
             "missing_total": 0,
             "missing_time_in_millis": 0,
```


图 2.8　查询索引文件返回的状态信息

在返回的结果中可以看到多个对象，其中，primaries 是当前节点之上的所有主分片的信息，total 是所有分片及副本的信息，indices 是当前操作的索引文件的信息。另外，这些对象都包含如下对象：

- docs：显示被索引文档的信息，其中的 count 值表示所描述的索引中的文档数量。

- store：反映索引的大小以及 throttle 信息等。
- indexing：索引操作信息。
- get：实时获取操作信息。
- search：搜索操作信息。

以上 JSON 对象是并列关系，结构相同。由于 get 对象内容过长，因此 search 对象未能容纳在图 2.8 中。事实上 search 对象是最后一个对象，排在 get 对象之后。

> Tips 查询时也可同时给出多个索引的统计信息，如“/索引 1，索引 2，索引 3/_stats”等。

24 通过映像配置索引

在关系型数据库管理系统中建立数据表时要定义其字段名及其数据类型，与此类似，映像(mappings)是对 Elasticsearch 中索引字段名及其数据类型的定义，但映像要比关系型数据库管理系统中的数据结构定义灵活得多。其实，在使用 Elasticsearch 时，不指定映像也是可以的(因为 Elasticsearch 会自动根据数据格式定义它的类型)，但如果需要对某些字段添加特殊属性(如该字段是否分词、是否存储、使用什么样的分析器等)，就必须添加和设置映像。在为索引文件添加映像时有两种方法，一种是定义在配置文件中，另一种是在运行时手动提交映像。

> Tips 映像默认可以为数据规定多种数据类型(type)。在 Elasticsearch 升级到 5.0 之后，去掉了原有的 string 类型，添加了新的 text 和 keyword 类型。其中，text 类型适用于电子邮件正文、产品描述等需要进行全文索引的信息，而 keyword 适用于电子邮箱地址、主机名、网络传输状态码、国际邮政编码、标记文本等能够精确命中字段的信息。

2.4.1 在索引中使用映像

代码段 2.5 就是手动提交映像的例子。代码运行后，会创建名为 weibo 的索引文件，其中有一个名为 user 的字段，其数据类型为 text 且设置为不对该字段内容进行分词。注意，在代码段 2.5 中，也同时指定了索引文件的 shards 数和 replicas 数。

代码段 2.5：通过 mappings 设置 index 中某个 type 下的 field 中的细节信息

```
curl -H 'Content-Type: application/json' -XPOST localhost:9200/weibo/_doc -d'
{
  "settings" : {
    "number_of_shards" : 5,
    "number_of_replicas" : 1
  },
  "mappings" : {
    "properties" : {
      "user" : {
            "type" : "text",
             "index" : false
        }
      }
    }
}'
```

Tips　在 Elasticsearch 5.x 及以前的版本中，每个索引中可以定义若干个类型。然而，类型在实际应用中存在一些问题，如不同名称的类型、同名字段的映像设置必须保持一致，这与关系型数据库中数据表的概念存在认知上的冲突。Elastic 公司计划在以后的大版本更新过程中移除类型的概念，在 Elasticsearch 6.0 及以后、7.0 之前的版本中，仅允许用户创建单一类型(例如上面例子中统一使用的_doc)。

2.4.2　管理/配置映像

一般地，可以使用下面的方法完成管理/配置映像：

```
PUT /{index}/_mapping
```

其中，PUT 代表 HTTP 方法，{index}表示对应的索引文件名称。

针对{index}，可以使用下面的语法进行扩充：

```
blank | * | _all | glob pattern | name1, name2, …
```

代码段 2.6 展示了管理/配置索引文件的映像的方法。此例中，在名为 weibo 的索引文件中，对名为 message 的字段进行了设置，这里定义 message 的数据类型为 text，并且需要在 Elasticsearch 中存储。

代码段 2.6：通过映像配置索引文件 weibo 的字段属性

```
curl -H 'Content-Type: application/json' -XPUT 'http://localhost:9200/weibo/
_doc/_mapping' -d '{
  "_doc" : {
    "properties" : {
      "message" : {
        "type" : "text",
          "store" : true
      }
    }
  }
}'
```

2.4.3 获取映像信息

可通过 GET 方法获取映像中的信息，相关命令为

```
GET /{index}/_mapping
```

下面的代码给出了从具体类型文件中获取信息的方法：

```
curl -H 'Content-Type: application/json' -XGET 'http://localhost:9200/weibo/
_mapping/_doc'
```

和 2.4.2 节所述情况类似，可在{index}中写入索引文件名称（如果有多个索引文件，可以使用逗号隔开，也可以使用_all 参数来匹配所有索引）。例如，如果要获取所有索引文件的所有类型文件的映像信息，可使用下面的方法：

```
curl -H 'Content-Type: application/json' -XGET 'http://localhost:9200/_all/
_mapping'
curl -H 'Content-Type: application/json' -XGET 'http://localhost:9200/_mapping'
```

代码段 2.7 演示了查看索引文件名为 weibo、字段为 user 的映像配置的方法。

代码段 2.7：查看指定映像配置的方法

```
curl -H 'Content-Type: application/json' -XGET 'http://localhost:9200/weibo/
_mapping/_doc/field/user'
```

上述语句的返回值如下：

```
{
  "myweibo3" : {
    "mappings" : {
      "_doc" : {
        "user" : {
          "full_name" : "user",
          "mapping" : {
            "user" : {
              "type" : "text",
              "fields" : {
                "keyword" : {
                  "type" : "keyword",
                  "ignore_above" : 256
                }
              }
            }
          }
        }
      }
    }
  }
}
```

代码段 2.8 演示了如何进行跨索引(即多个索引)查询映像信息的方法,以及在所有索引中针对某些指定字段内容的查询方法(注意其中通配符的用法)。

代码段 2.8: 在多个索引或多个类型中查询映像信息的方法

```
curl -H 'Content-Type: application/json' -XGET 'http://localhost:9200/weibo,weibo2/_mapping/field/
time'
curl -H 'Content-Type: application/json' -XGET 'http://localhost:9200/_all/
_mapping/_doc/field/content,time,message'
curl -H 'Content-Type: application/json' -XGET 'http://localhost:9200/_all/
_mapping/wei*/field/*.id'
```

2.4.4 删除映像

映像可以通过 DELETE 方法删除。这里也允许使用通配符和_all 等参数。注意,在下面给出的方法中,使用的 HTTP 方法是 DELETE 方法。

```
DELETE /{index}
DELETE /{index}/_mapping
```

在上述语句后面的{index}参数列表中可使用的参数如下(如果存在多个名称,用逗号分隔开即可):

```
* | _all | glob pattern | name1, name2, …
```

25 管理索引文件

2.5.1 打开、关闭、检测、删除索引文件

一个关闭的索引文件将禁止用户向其写入数据或读取其中的数据,而一个打开的索引文件可以允许用户对其中的数据文件进行相应操作。使用 /{index 名称}/_open 和 /{index 名称}/_close 语句可以打开或关闭指定的索引文件,具体方法见代码段 2.9。

代码段 2.9: 打开及关闭索引文件 weibo 的方法

```
curl -H 'Content-Type: application/json' -XPOST 'localhost:9200/weibo/_open'
curl -H 'Content-Type: application/json' -XPOST 'localhost:9200/weibo/_close'
```

通过 HEAD 方法可以检测一个索引文件是否存在。代码段 2.10 检测索引文件 weibo 是否存在(通过返回的 HTTP 状态码进行检查)。如果返回的状态码是 200,则表示存在相应的索引文件;如果返回的状态码是 404,则表示文件不存在。该命令中的-XHEAD 在新版本中推荐使用-I 或--head 代替。

代码段 2.10: 检测索引文件 weibo 的状态

```
curl -H 'Content-Type: application/json' -XHEAD 'http://localhost:9200/weibo' -v
```

类似地,通过 DELETE 方法可以删除一个或者多个索引文件。例如,代码段 2.11 删除名为 weibo1 的索引文件。

代码段 2.11: 删除索引文件 weibo1

```
curl -H 'Content-Type: application/json' -XDELETE 'http://localhost:9200/weibo1/'
```

还可使用通配符来批量删除名称相似的索引文件。例如,代码段 2.12 删除以 weibo 字符串开头的一组索引文件。同样,也可使用_all 参数删除全部索引文件。

代码段 2.12：使用通配符批量删除索引文件

```
curl -H 'Content-Type: application/json' -XDELETE 'http://localhost:9200/
weibo*/'
```

2.5.2 清空索引缓存

通过下面的方法，可以清空指定的索引缓存，参见代码段 2.13。

代码段 2.13：清空 weibo 中的索引缓存

```
curl -H 'Content-Type: application/json' -XPOST 'http://localhost:9200/weibo/_cache/clear'
```

如果指定了多个索引文件名称，也可以一次清空多个索引缓存。例如，代码段 2.14 所示的方法可以清空两个索引文件 weibo1、weibo2 中的缓存。

代码段 2.14：清空多个索引缓存

```
curl -H 'Content-Type: application/json' -XPOST 'http://localhost:9200/
weibo1,weibo2/_cache/clear'
```

2.5.3 刷新索引文件

通过代码段 2.15 的方法，可以刷新一个或多个索引文件的状态，以便反映索引文件的最新变化。

代码段 2.15：刷新索引文件

```
curl -H 'Content-Type: application/json' -XPOST 'http://localhost:9200/weibo/_refresh'
```

也可以通过指定多个索引名称一次刷新多个索引文件（分别指定索引文件名，如代码段 2.16第一行代码所示）或全部索引（无须指出索引文件名，如代码段 2.16 第二行代码所示）。

代码段 2.16：一次刷新多个索引文件或全部索引文件

```
curl -H 'Content-Type: application/json' -XPOST 'http://localhost:9200/
weibo1,weibo2/_refresh'
curl -H 'Content-Type: application/json' -XPOST 'http://localhost:9200/
_refresh'
```

2.5.4 优化索引文件

相对于 Lucene 的索引，Elasticsearch 的索引过程多了分布式数据的扩展，它主要使用 translog 进行各节点间的数据平衡。例如，代码段 2.17 可以优化一个或多个索引文件。

代码段 2.17：优化索引文件

```
curl -H 'Content-Type: application/json' -XPOST 'http://localhost:9200/weibo/_doc/_optimize' -d {}
```

类似地，也可以指定多个索引文件名称，完成一次优化多个索引文件或全部索引的任务，其方法同代码段 2.16，不再赘述。

2.5.5 flush 操作

flush 操作可将暂存于内存中的临时数据送至指定索引文件并清空内部操作日志等，方法如代码段 2.18 所示。

代码段 2.18：flush 操作

```
curl -H 'Content-Type: application/json' -XPOST 'http://localhost:9200/weibo/_flush'
```

类似地，也可以指定多个索引文件名称，完成一次更新多个索引或全部索引的任务，其方法同代码段 2.16，不再赘述。

26 设置中文分析器

全文检索往往需要对中文进行分词。有关中文分词的背景知识可参阅 2.9 节。Lucene 提供了中文分析器用于处理中文分词，此外也有一些开源的中文分词处理模块可以方便地集成到信息检索系统中。如果需要为当前的索引文件定义一个新的中文分析器，需要先关闭当前索引文件，然后更改中文分析器，最后再次打开这个索引文件。这里以 IK 插件为例，对中文分析器的配置和使用进行介绍。

首先，在 IK 的 GitHub 网站页面中（网址：https://github.com/medcl/elasticsearch-analysis-ik）下载源代码压缩包。单击页面左上方标有“branch：master”字样的下拉列表，单击 tags 标签，选择与当前已安装的 Elasticsearch 相对应的版本（可以在页面下方的对照表中查看应下载的 IK 分析器版本），页面随即跳转至指定版本的源代码页面。此时单击右上方的绿色按钮，下载 ZIP 压缩包。

> Tips IK 的源代码可以使用 Maven 编译。也可以在 http://maven.aliyun.com 找到 IK 的已编译程序。此外，IK 插件的 GitHub 页面中也给出了直接下载已编译程序包(pre-build package)的资源链接。

接着，使用终端中的 unzip 命令将下载的压缩包解压。在解压后的文件夹中，执行终端命令 mvn package 生成 IK 分词插件的 ZIP 压缩包。在 Elasticsearch 目录下的 plugins 文件夹中创建 ik 文件夹，并将生成的 ZIP 压缩包解压到该文件夹中。然后对 Elasticsearch 中的索引文件 weibo 进行更改中文分析器的配置，如代码段 2.19 所示。

代码段 2.19：设置索引文件的中文分析器

```
curl -H 'Content-Type: application/json' -XPOST 'localhost:9200/weibo/_close'          //关闭 weibo
curl -H 'Content-Type: application/json' -XPUT 'localhost:9200/weibo/_settings' -d '{    //设置 weibo
  "analysis": {                                   //更改中文分析器
    "analyzer": {
      "content": {                                //设置应用中文分析器的字段
        "type": "custom",
        "tokenizer": "ik_max_word"                //采用 IK 中文分析器
      }
    }
  }
}'
curl -H 'Content-Type: application/json' -XPOST 'localhost:9200/weibo/_open'      //重新打开 weibo
```

代码段 2.20 演示了通过特定的中文分析器对指定文字进行分词并观察分词结果的方法。

代码段 2.20：使用 IK 中文分析器对指定文字进行分词

```
curl -H 'Content-Type: application/json' -XGET "http://localhost:9200/weibo/
_analyze" -d'{
   "text":"公安部：各地校车将享最高路权","tokenizer": "ik_smart"
}'
```

上述代码执行后，ik_smart 分析器将句子以最粗粒度拆分，该示例的返回值如下：

```
{
  "tokens": [
    {
      "token" : "公安部",          #第 1个词
      "start_offset" : 0,
      "end_offset" : 3,
      "type" : "CN_WORD",
      "position" : 0
    },
    {
      "token" : "各地",            #第 2个词
      "start_offset" : 4,
      "end_offset" : 6,
      "type" : "CN_WORD",
      "position" : 1
    },
    {
      "token" : "校车",            #第 3个词
      "start_offset" : 6,
      "end_offset" : 8,
      "type" : "CN_WORD",
      "position" : 2
    },
    {
      "token" : "将",              #第 4个词
      "start_offset" : 8,
      "end_offset" : 9,
      "type" : "CN_CHAR",
      "position" : 3
    },
    {
      "token" : "享",              #第 5个词
      "start_offset" : 9,
      "end_offset" : 10,
      "type" : "CN_CHAR",
      "position" : 4
    },
    {
      "token" : "最高",            #第 6个词
```

```
        "start_offset" : 10,
        "end_offset" : 12,
        "type" : "CN_WORD",
        "position" : 5
      },
      {
        "token" : "路",              #第 7个词
        "start_offset" : 12,
        "end_offset" : 13,
        "type" : "CN_CHAR",
        "position" : 6
      },
      {
        "token" : "权",              #第 8个词
        "start_offset" : 13,
        "end_offset" : 14,
        "type" : "CN_CHAR",
        "position" : 7
      }
    ]
  }
```

27 对文档的其他操作

Elasticsearch 提供了多种途径对文档进行获取信息、增删改、更新等相关操作。

2.7.1 获取指定文档的信息

可以通过 GET 方法获取指定文档的详细信息。代码段 2.21 演示了查看索引文件 myweibo3 下 id 号为 3 的文档记录信息的方法，注意这里的 HTTP 操作方式是 GET。

代码段 2.21：获取指定文档的信息

```
curl -H 'Content-Type:application/json' -XGET 'http://localhost:9200/
myweibo3/_doc/3'
```

示例返回结果如下，其中，_source 段中的内容就是指定文档的详细信息。

```
{
    "_index" : "myweibo3",                  //索引名
    "_type" : "_doc",                       //类型名
    "_id" : "3",                            //id号
    "_version" : 1,                         //版本号
    "found" : true,
    "_source" : {
        "user": "LiMing",                   //详细信息
        "post_date": "2018-01-01T14:00:00",
        "mymessage": "Hello Tom"
    }
}
```

除了上面的方法外，还可设置想要显示或屏蔽的结果。代码段 2.22 演示了关闭_source过滤器后的效果（语句中的问号表示其后是参数，问号后的 pretty 表示返回的结果以缩进格式显示以方便阅读）。执行该语句后，在输出的结果中将不再带有详细信息。

代码段 2.22：_source 过滤器

```
curl -H 'Content-Type:application/json' -XGET 'http://localhost:9200/
myweibo3/_doc/3?pretty&_source=false'
```

代码段 2.23 演示了如何只显示特定字段的方法（此例只显示_source 为 user 的内容）。

代码段 2.23：显示特定字段的方法

```
curl -H 'Content-Type:application/json' -XGET 'http://localhost:9200/myweibo3/_doc/3?_source=user'
```

上述语句的返回结果将只显示有 user 字段，如下所示：

```
{
    "_index" : "myweibo3",
    "_type" : "_doc",
    "_id" : "3",
    "_version" : 1,
    "found" : true,
    "_source": {
        "user": "LiMing"
    }
}
```

2.7.2　删除指定文档的信息

可以使用 DELETE 方法删除指定文档的信息。例如，执行代码段 2.24，会删除 myweibo3 索引文件中 id 号为 3 的文档的信息。

代码段 2.24：删除指定文档的信息

```
curl -H 'Content-Type:application/json' -XDELETE 'http://localhost:9200/
myweibo3/_doc/3'
```

上述语句的返回结果如下，表明删除成功。

```
{
  "found": true,
  "_index": "myweibo3",
  "_type": "_doc",
  "_id": "3",
  "_version": 2,
  "result": "deleted",
  "_shards": {
    "total": 2,
    "successful": 2,
    "failed": 0
  }
}
```

2.7.3　更新指定文档的信息

如果需要对索引文件中的类型或其中的文档进行更新，需要用 Elasticsearch 提供的 update API 来实现。它的处理过程是：先取出文档，运行指定脚本，然后更新文档。例如，代码段 2.25 是在索引文件 weibo 中增加 id 为 3 的文档的信息（例子中给出了 4 个字段，即 user、post_date、message、like，这里的 like 的含义是针对此条微博的点赞数量）。

代码段 2.25：直接向索引文件中增加指定 id 号的文档的信息

```
curl -H 'Content-Type:application/json' -XPUT http://localhost:9200/weibo/
_doc/3 -d '{     //直接指定 id 号，这里的 HTTP 方法是 PUT
  "user": "LiMing",
  "post_date": "2018-01-10T14:00:00",
  "message": "Hello Tim",
  "like": 3
}'
```

接下来，使用 script 定制函数形式的命令对文档信息进行更新。可以通过使用 update API 将上述文档的 like 数值增大，实现方法如代码段 2.26 所示。注意，这里的 ctx._source 表示本文档的内容，ctx._source.like 表示本文档中某个具体的字段（这里指 like 字段），params 是拟使用的新参数值。

代码段 2.26：通过 update API 更新文档信息

```
curl -H 'Content-Type:application/json' -XPOST http://localhost:9200/weibo/
_doc/3/_update -d '{
    "script":{
        "inline":"ctx._source.like+=params.count",
        "lang":"painless",
        "params":{
            "count":4
        }
    }
}
```

执行完上述语句后的索引数据如图 2.9 所示。注意，字段中的 like 值已经由原来的 3 变为 7。

```
{
    "_index": "weibo",
    "_type": "_doc",
    "_id": "3",
    "_version": 2,
    "_score": 1,
    "_source": {
        "user": "Liming",
        "post_date": "2018-01-10T14:00:00",
        "message": "Hello Tom",
        "like": 7
    }
}
```

图 2.9　更新 like 字段后的索引数据

Tips 上面的代码中，ctx._source 表示文档内容，ctx._source.like 表示该文档的 like 字段，params 中的 count 用于为变量赋值。图 2.9 中的结果可以通过 curl -H 'Content-Type: application/json' -XGET http://localhost:9200/weibo/_doc/3 命令得到。

上述的 update API 是针对数值型数据的增删改操作。类似地，也可以完成对字符型数据的更新。代码段 2.27 录入了 id 号为 5、索引文件为 weibo 的部分微博数据字段（含 user、

post_date、message、tags 等）。

代码段 2.27：直接录入指定 id号的文档的信息

```
curl -H 'Content-Type: application/json' -XPOST http://localhost:9200/weibo/
_doc/5 -d '{    //注意这里使用的 HTTP 方法是 POST
    "user": "LiMing",
    "post_date": "2016-11-02T14:00:00",
    "message": "Hello Lily",
    "script.engine.groovy.inline.update": "true",
    "script.inline": "true",
    "script.stored": "true",
    "tags": [
        "Hello"
        ]
    }'
```

代码段 2.28 的作用是修改上述语句执行结果中的 tag 字段信息并增加新的内容（即在 tag 字段中增加新的内容）。修改后的文档信息如图 2.10 所示（可通过 curl -H 'Content-Type：application/json' -XGET http://localhost:9200/weibo/_doc/5 命令得到运行结果）。

```
"_source": {
    "user": "LiMing",
    "post_date": "2018-01-15T14:00:00",
    "message": "Hello Lily",
    "script.engine.groovy.inline.update": "true",
    "script.inline": "true",
    "script.stored": "true",
    "tags": [
        "Hello"
        ,
        "Today is a nice day!"
    ]
}
```

图 2.10　更新 tag 字段后的索引数据

代码段 2.28：通过 update API 更新文档信息

```
curl -H 'Content-Type: application/json' -XPOST http://localhost:9200/weibo/
_doc/5/_update -d '{ //注意这里的 HTTP 方法是 POST
    "script": {
        "inline": "ctx._source.tags.add(params.tag)",
        "lang": "painless",
        "params": {
            "tag": "Today is a nice day!"
        }
    }
}'
```

由于 Elasticsearch 处理的文档多是非结构化的信息，因此它可以与关系型数据库不同，文档中各记录的字段有可能不一样。利用 update API，不仅可以用上述方法更新数据，也可以修改文档结构（如只在某一 id 号的文档的记录上增加新字段）。代码段 2.29 在指定文档中增加新字段，这个新字段的名字是 name_of_new_field，其值是 new_field。注意，这里对引号需作转义处理。

代码段 2.29：通过 update API 修改文档结构并增加新的字段

```
curl -H 'Content-Type: application/json' -XPOST http://localhost:9200/weibo/
_doc/5/_update -d '{
  "script": "ctx._source.name_of_new_field=\"new_field\""
}'
```

代码段 2.30 完成的任务如下：在使用 update API 时，如果指定 id 号（此例中是 7）的记录不存在，则通过参数中的 upsert 创建这个文档，并且加入新的字段（其名为 counter，其值为 1）；如果有该记录，就把指定的字段 like 的值增加 4（在代码中的 params 中给出要增加的值）。

代码段 2.30：通过 update API 修改文档结构

```
curl -H 'Content-Type: application/json' -XPOST http://localhost:9200/weibo/
_doc/7/_update -d '{
    "script": {
        "inline": "ctx._source.like+=params.count",
        "lang":"painless",
        "params": {
            "count": 4
        }
    },
    "upsert": {
        "counter": 1
    }
}'
```

2.7.4 基于 POST 方式批量获取文档信息

Elasticsearch 支持一次操作多条记录。例如，代码段 2.31 可返回 id 号为 5 和 7 的文档信息，注意语句中的_mget 参数的使用。

代码段 2.31：基于 POST 方式批量获取文档信息

```
curl -H 'Content-Type: application/json'-XPOST http://localhost:9200/weibo/
_mget -d '{
  "docs": [
    {
      "_index": "weibo",
      "_id": "5"
    },
    {
      "_index": "weibo",
      "_id": "7"
    }
  ]
}'
```

执行代码段 2.31 的返回结果如图 2.11 所示。

```
{
  ▼ "docs": [
      {
        "_index": "weibo",
        "_type": "_doc",
        "_id": "5",
        "_version": 3,
        "found": true,
      ▼ "_source": {
          "user": "LiMing",
          "post_date": "2018-01-15T14:00:00",
          "message": "Hello Lily",
          "script.engine.groovy.inline.update": "true",
          "script.inline": "true",
          "script.stored": "true",
        ▼ "tags": [
            "Hello"
            ,
            "Today is a nice day!"
          ],
          "name_of_new_field": "new_field"
        }
      }
      ,
      {
        "_index": "weibo",
        "_type": "_doc",
        "_id": "7",
        "_version": 1,
        "found": true,
      ▼ "_source": {
          "counter": 1
        }
      }
  ]
}
```

图 2.11　基于 POST 方式批量获取文档信息

也可使用_source 过滤器。代码段 2.32 演示了相应的方法。

```
代码段 2.32：使用_source 过滤器获取文档信息
curl -H 'Content-Type: application/json' -XPOST curl 'localhost:9200/_mget?
pretty' -d '{
  "docs" : [
    {
      "_index" : "weibo",
      "_id" : "3",
      "_source": false
    }
    ]
}'
```

针对上述语句的返回结果如下：

```
{
  "docs" : [
    {
      "_index" : "weibo",
      "_type" : "_doc",
      "_id" : "3",
      "_version" : 2,
      "found" : true,
      "_source" : {
        "user" : "Liming",
        "post_date" : "2018-01-10T14:00:00",
        "message" : "Hello Tom",
        "like" : 7
      }
    }
  ]
}
```

28 实例

前面给出了如何建立索引文件、通过映像配置索引文件和管理索引文件、为文档设置中文分析器以及对文档的其他相关操作方法。下面给出一个在 Elasticsearch 中建立索引文

档的实例,即对获取的非结构化的学术论文数据建立基于 Elasticsearch 的索引文件的方法。实例中的数据是来自网络的图书数据,包含图书的书名、作者、内容简介和关键字等信息,并对它们建立索引。下面介绍如何建立基于 Elasticsearch 的索引文件。

首先,建立一个名为 information 的索引文件,该索引文件中的性能指标(如 number of shards 和 number of replicas)均采用默认值。

其次,为 information 索引文件创建映像。在创建传统的关系型数据库中的数据表时,一般要逐个创建数据表中的每个字段并指定其数据类型。在 Elasticsearch 中,虽然它会为拟创建的索引文件自动创建映像,但在实际使用时,通常要根据实际情况手动创建映像,这样可方便进行个性化设置(如规定数据类型、在某字段上执行各种分词操作等)。图 2.12 是借助 head 工具,给出针对本实例的手动配置索引映像的结果。从中可见,在 information 索引中包括 4 个字段,其中 abstract 和 keywords 采用 ik_max_word 分析器。

▼查询
http://localhost:9200/
information　PUT

```
{
  "mappings": {
    "_doc": {
      "properties": {
        "title": {
          "type": "text",
          "index": true,
          "analyzer": "ik_max_word"
        },
        "abstract": {
          "type": "text",
          "index": true,
          "analyzer": "ik_max_word"
        },
        "author": {
          "type": "text",
          "index": false
        },
        "keywords": {
          "type": "text",
          "index": true,
          "analyzer": "ik_max_word"
        }
      }
    }
  }
}
```

提交请求　验证 JSON　易读

图 2.12　创建映像

> Tips 在 Elasticsearch 5. x 及后续版本中，应使用 ik_smart 或 ik_max_word 作为 IK 分词工具的名称。其中，ik_smart 会做最粗粒度的拆分，例如会将“中华人民共和国国歌”拆分为“中华人民共和国/国歌”；而 ik_max_word 会将文本做最细粒度的拆分，例如会将“中华人民共和国国歌”拆分为“中华人民共和国/中华人民/中华/华人/人民共和国/人民/人/民/共和国/共和/和/国国/国歌”。

创建映像后，可查看索引文件中任意字段的映像配置信息。图 2.13 显示的是索引文件 information 中字段为 author 的映像配置信息。

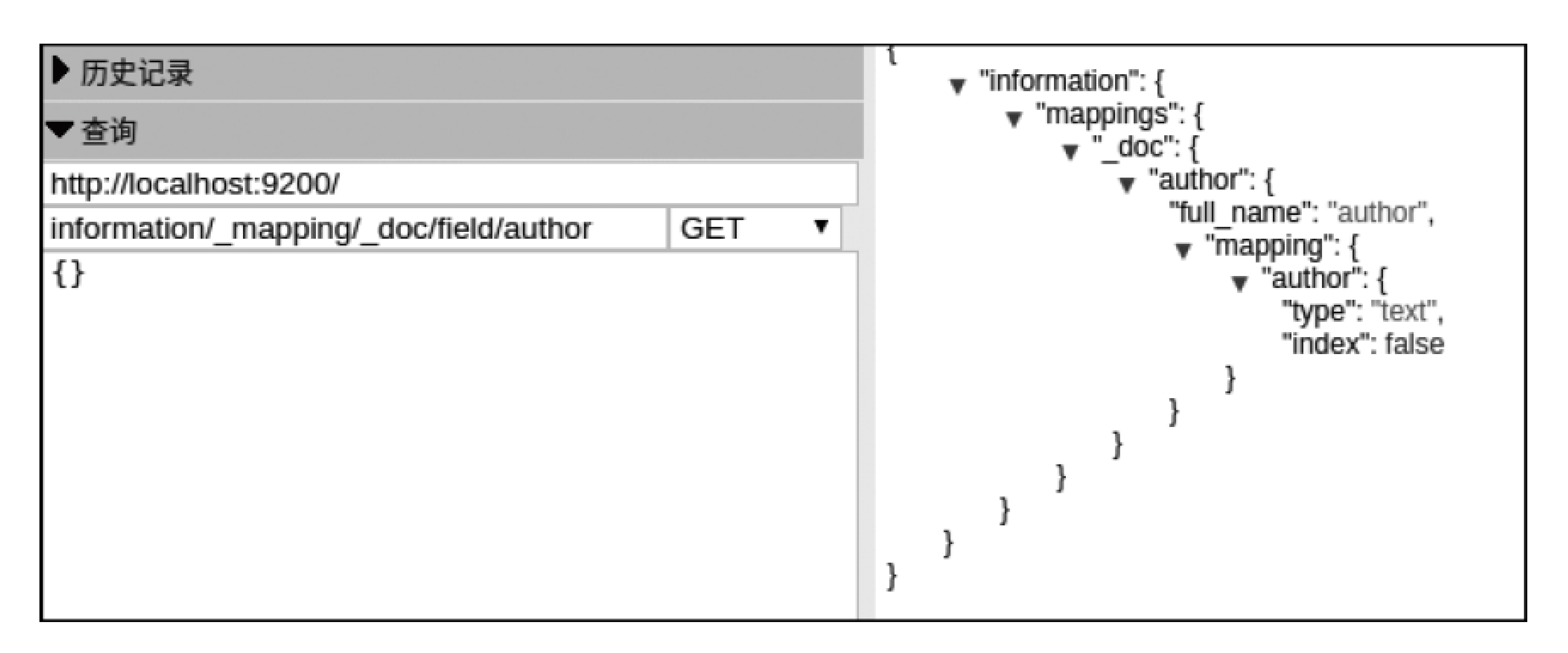


图 2.13　查看指定字段的映像配置信息

第三，录入图书数据。可使用两种方法：一种是在 head 中手动录入数据（如图 2.14 所示）；另一种是通过 Java 客户端方式自动实现。由于此处尚未介绍 Java 客户端的相关方法，这里给出前一种方法。图 2.15 为添加好的相关数据。

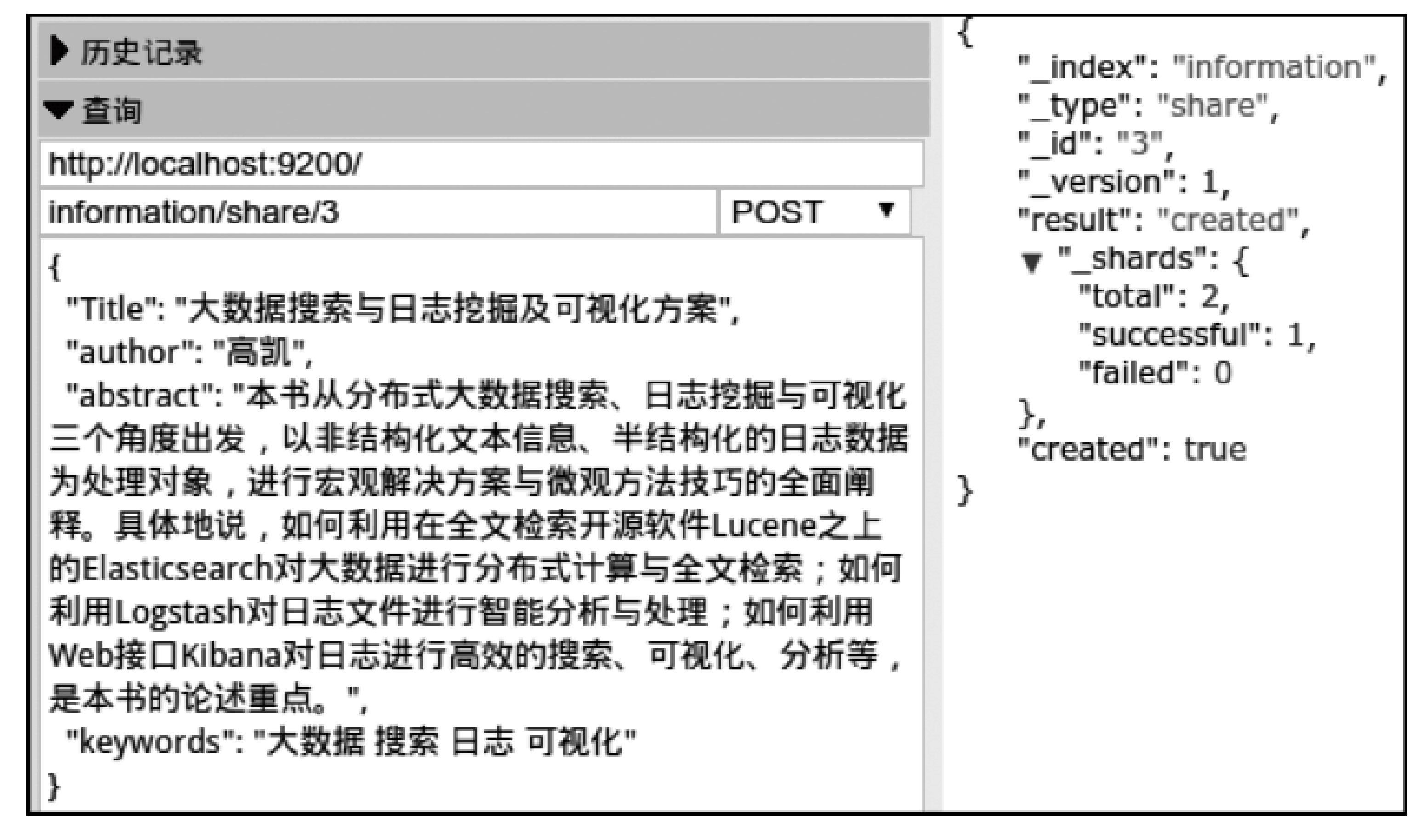


图 2.14　录入信息

_id	_score ▲	title	author	abstract
5	1	TensorFlow：实战Google深度学习框架（第2版）	郑泽宇 梁博文 顾思宇	TensorFlow是谷歌2015年开
2	1	机器学习	周志华	机器学习是计算机科学与人
4	1	数学之美（第二版）	吴军	《数学之美》上市后深受广
1	1	深度学习	Ian Goodfellow，Yoshua Bengio，Aaron Courville	《深度学习》由全球知名的
3	1	大数据搜索与挖掘及可视化管理方案（第3版）	高凯	本书着重介绍关于Elastic S

图 2.15　存储到 Elasticsearch 中的信息

当把相关信息存储到文档中以后，可以对文档进行相关操作（如获取指定的文档信息、获取文档信息中指定的字段、删除部分信息、数据更新、批量获取文档等）。图 2.16 为获取 id 为 3 的文档信息（通过 HTTP 的 GET 方式实现）。图 2.17 为获取 id 为 1 的文档的 _source下的 title（通过 HTTP 的 GET 方式实现）。图 2.18 为删除 id 为 2 的文档信息（通过 HTTP 的 DELETE 方式实现）。图 2.19 为基于 POST 方式批量获取文档信息（通过 HTTP 的 POST 方式实现）。

▶ 历史记录
▼ 查询
http://localhost:9200/
information/_doc/3　GET
{}

```
{
    "_index": "information",
    "_type": "_doc",
    "_id": "3",
    "_version": 1,
    "found": true,
    ▼ "_source": {
        "title": "大数据搜索与挖掘及可视化管理方案（第3版）",
        "author": "高凯",
        "abstract": "本书着重介绍关于Elastic Stack的基本知识，带领读者深入了
        读者掌握相关的大数据搜索、日志分析和可视化技术。本书还详细介绍了
        "keywords": "大数据 搜索 日志 可视化"
    }
}
```

图 2.16　获取指定 id 的文档信息

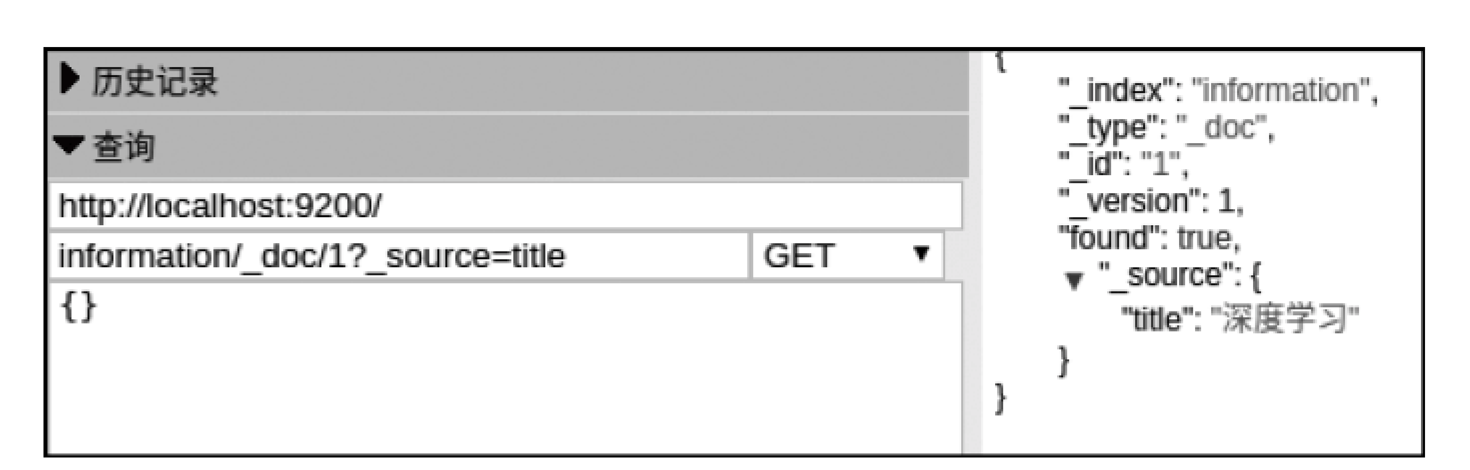


图 2.17　获取指定字段 title 的信息

▶ 历史记录
▼ 查询
http://localhost:9200/
information/_doc/4　DELETE
{}

```
{
    "_index": "information",
    "_type": "_doc",
    "_id": "4",
    "_version": 2,
    "result": "deleted",
    ▼ "_shards": {
        "total": 2,
        "successful": 1,
        "failed": 0
    },
    "_seq_no": 2,
    "_primary_term": 2
}
```

图 2.18　删除指定 id 的文档信息



图 2.19　基于 POST 方式批量获取文档信息

29　扩展知识与阅读

实现人机间自然语言的交互，意味着要使计算机既能理解自然语言的意义，也能以自然语言文本来表达给定的意图、思想等。前者一般称为自然语言理解或自然语言处理（Natural Language Processing，NLP），后者是自然语言生成。NLP 包括词法分析（涉及自动分词、分词歧义消解、未登录词识别与获取等）、语法分析（涉及自动标注、语法表示、语法分析等）、语义分析（涉及语义表示、语义分析、语义消歧等），其应用包括文本分类、信息抽取、自动文摘、统计对齐与机器翻译、聚类等。在信息检索中，自然语言包括关键词、自由词和出现在文献题名、摘要、正文或参考文献中的具有一定实际意义的词语。研究自然语言处理，是希望计算机能够在某种程度上理解并处理人类的自然语言。目前各种自然语言处理技术都相继出现并已应用到某些领域（张华平，2014）。

国内外关于自然语言处理的研究曾长期专注于语法层次。20 世纪末，人们认识到，单纯从语法层次上进行研究是不能解决实际问题的。目前，国内外在信息检索领域内对自然语言理解的研究已有很多成果。研究方法有基于语言学规则分析的方法和基于统计的方法两种（高凯，2014）。基于规则分析的方法是建立形式化的知识系统来阐述语言知识，规则多

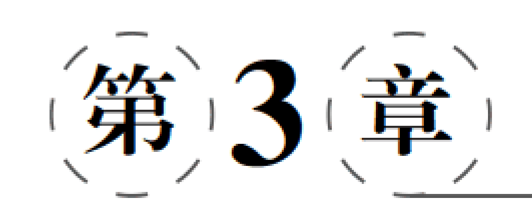
是通过内省的方式得到的,其本质是一种确定性的演绎推理。虽然这种方法可以实现对单个句子的分析,但是难以覆盖各种复杂的语言现象。同时,如果要添加新的规则,还需要注意到与已有规则间的相容性。正因为基于规则的方法的诸多缺陷的存在,再加之后来大量语料库的涌现,使得基于统计的方法逐渐兴起,其最大的优点是可以使语言现象数量化,再加上其他的一些优势,使这种方法的应用范围非常广泛。但是该方法可能会掩盖一些小概率事件的发生,所以该方法也有局限性。总之,两种方法各有优缺点。将二者结合起来,优势互补,并借此实现面向大规模真实文本的信息处理,是可行的。

在进行自然语言处理时,分词——特别是对像中文这样的亚洲语言来说——是必要的。相对于以空格自然分开的英文来说,中文分词处理要复杂得多,因为在英文中广泛存在的空格就是最简单的分词标志(两个空格之间的字符串即被定义为一个 token),但是对中文而言,问题就没有这么简单了。例如中文中广泛存在着的切分歧义等问题,在西文中就基本没有。中文的切分歧义包括交叉歧义(如“乒乓球拍卖完了”等)、组合歧义(需要根据上下文甚至整个句子来判断)、真歧义(由人来判断也不知道应该怎样切分合适)等。又如,中文信息处理中对未登录词的处理也是一个难点。由于全文检索需要对没有分割标记的文本进行分析和处理,因此基本的分词处理是必要的,特别是在对文本数据建立倒排索引时。

倒排索引是一种数据结构,常见的搜索引擎系统多采用词作为检索项,可检索某个词,找到这个词出现的所有文档及出现位置。简单地说,倒排索引类似一个字典,该字典中的每一个条目都指向一个列表,列表中的每一项指向一个此条目在某个文档中出现的位置。这个条目的值应该是词,而不是单独的字。例如,要查询出现“计算机”3 个字的文档,当然不希望查询的结果中出现“计”或“算”或“机”这 3 个单个的字,因为“计算机”是一个具有单独意思的词,所以分词的准确度直接影响搜索引擎的可用性。一般认为,对文档的分词处理是对其进行分类等处理的前期步骤。

2.10　本章小结

本章主要介绍了 Elasticsearch 中和索引相关的知识,内容包括基于 RESTful 的接口进行文档索引的方法,内容涉及索引建立、文档操作等。Elasticsearch 的映像类似于静态语言中的数据类型。映像不仅告诉 Elasticsearch 一个字段中存储的是什么类型的值,还告诉 Elasticsearch 如何索引数据以及数据以何种形式被搜索到。



信息检索与聚合

> Elasticsearch uses Lucene under the covers to provide the most powerful full text search capabilities available in any open source product. Search comes with multi-language support, a powerful query language, support for geolocation, context aware did-you-mean suggestions, autocomplete and search snippets. Elasticsearch provides a full Query DSL based on JSON to define queries. Query clauses behave differently depending on whether they are used in query context or filter context. The aggregations framework helps provide aggregated data based on a search query. It is based on simple building blocks called aggregations, that can be composed in order to build complex summaries of the data. An aggregation can be seen as a unit-of-work that builds analytic information over a set of documents.
>
> http://www.elastic.co

在 Elasticsearch 中的基于 RESTful 接口方式中，完成信息检索功能的关键词是_search，通过 POST 的方式发送到 Elasticsearch，其后再跟“?q=查询词”等，其形式化表示方式是

```
http://{ip_address}:{port}/{index}/{type}/_search?q=查询词
```

例如，可以以 curl -XGET 'http://localhost:9200/_search?q=hello+world'的方式完成简单检索。但是，Elasticsearch 的信息检索功能很强大，其功能远远不止于此。在进行检索时，可简单地使用基于 Lucene 的通用检索语法，也可以使用更加灵活的基于 JSON 格式的 Query DSL(Domain Special Language，领域专用语言)来构造各种检索请求，同时可以使用 Aggregations 来实现聚合。在一般的检索表达式中，有时也可同时包含查询条件和过滤器，可以使用复合查询，可以改变查询结果的排序，也可以在 Elasticsearch 中使用脚本。在

处理字段、类型和查询时可以指定分析器。和 Lucene 类似，Elasticsearch 索引和检索时使用的分析器(带有零个或多个过滤器的分词器，如中文分词工具)应该是一样的，否则可能导致检索失败。

3.1 实验数据集描述

首先介绍在示例中可能用到的 3 个主要数据文件的结构：

(1) 索引文件 baidu 下的类型文件 baike：利用爬虫采集到的百度百科词条信息，其数据结构如下：

```
_index: baidu                   //针对百度百科词条数据的索引文件名称
_type: baike                    //针对百度百科词条数据的类型文件名称
_id: ×××                        //id 号
_version: ×                     //版本号
_score: ×                       //排序分值
_source: {                      //数据字段描述
    title: ×××                  //词条标题
    url: ×××                    //URL,如 http://baike.baidu.com/view/6505879.htm
    content: ×××                //词条内容
    lastModifyTime: ×××         //最近更新时间
    taglist: ×××                //内容分类,如"历史人物"等
}
```

(2) 索引文件 it-home 下的类型文件 posts：利用爬虫采集到的程序员论坛主题帖信息，其结构如下：

```
_index: it-home                 //针对程序员论坛主题帖数据的索引文件名称
_type: posts                    //针对程序员论坛主题帖数据的类型文件名称
_id: ×××                        //id 号
_version: ×                     //版本号
_score: ×                       //排序分值
_source: {                      //数据字段描述
    publishTime: ×××            //帖子发表时间
    category: ×××               //主题类别
    title: ×××                  //帖子主题
    user: ×××                   //帖子发布者昵称
    url: ×××                    //URL,如 http://baike.baidu.com/view/6505879.htm
    content: ×××                //帖子内容
}
```

(3) 索引文件 whale 下的类型文件 log：为日志信息，其结构如下：

```
_index: whale                          //索引文件名称
_type: log                             //类型文件名称
_id: ×××                               //id号
_version: ×                            //版本号
_score: ×                              //排序分值
_source: {                             //数据字段描述
    custom_ip: ×××                     //客户端 IP 地址
    timestamp: ×××                     //时间戳
    http_method: ×××                   //HTTP 方法,如 GET、POST 等
    uri: ×××                           //请求 URI 标识
    status_code: ×××                   //网络状态码
    os: ×××                            //客户端使用的操作系统,如 Windows 10
    log_size: ×××                      //当次日志长度
}
```

3.2 基本检索

如果需要构造一个简单的查询语句(含对结果排序、控制返回数据集大小、指定返回字段等)，可以使用 Elasticsearch 提供的基本检索功能。

3.2.1 检索方式

Elasticsearch 的检索可以通过在浏览器地址栏中输入 URL、使用终端 curl 命令、在可视化工具 head 的复合查询中输入查询语句等方式进行(可任选一种方式进行查询)。

在 Elasticsearch 启动的前提下，可以在浏览器中直接输入 URL(如果同时提供 &pretty=true 子句，则输出 JSON 格式的结果是有缩进的)，格式如下：

```
http://localhost:9200/{index}/{type}/_search?q={field}:{keyword}&pretty=true
```

上述方法是可以在指定的索引文件 index 下的类型文件 type 的字段 field 中查找包含 keyword 字符串的结果集。可以为查询指定索引文件名和类型文件名(但也不是必需的)。如果只给出索引文件名而没有指定类型文件名，则检索该索引文件下的所有类型文件。也可以指定多个索引文件，查询其中的所有或特定的类型文件。下面列出几种不同的情况：

(1) 查询指定索引文件和指定类型文件下的信息(指定 index 和 type)的示例如下：

```
curl -XGET 'localhost:9200/it-home/posts/_search?q=category:java&pretty=true'
```

(2) 查询指定索引文件下所有类型文件中的信息(指定 index,不指定 type)的示例如下：

```
curl -XGET 'localhost:9200/it-home/_search?q=category:java&pretty=true'
```

(3) 查询所有索引文件中的信息(不指定 index 和 type)的示例如下：

```
curl -XGET 'localhost:9200/_search?q=content:java&pretty=true'
```

(4) 查询多个索引文件下所有类型文件中的信息(指定多个 index,不指定 type)的示例如下：

```
curl -XGET 'localhost:9200/baidu, it-home/_search?q=content:java&pretty=true'
```

(5) 查询多个索引文件下多个类型文件中的信息(指定多个 index 和多个 type)的示例如下：

```
curl -XGET 'localhost:9200/baidu,it-home/baike,posts/_search?q=content:java&pretty=true'
```

> Tips 在上面第(5)点中,虽可查询出相关数据,但由于 Elasticsearch 6.0 以上版本已不支持在索引文件中存放多个类型文件,因此这里的查询多类型信息,是指每个索引文件下的单个类型名。例如索引文件 baidu 下只有类型文件 baike,索引文件 it-home 下只有类型文件 posts,在查询时可以指定索引文件为 baidu、it-home,指定类型文件为 baike、posts,从而实现多个索引文件和多个类型文件查询。

3.2.2 query 查询

利用查询语句进行查询时,在查询体中使用 query 语句,即可执行对现有存储数据的查询。代码段 3.1 是使用 query 语句查询所有数据的示例。

代码段 3.1:使用 query 语句查询所有数据

```
curl -H 'Content-Type: application/json' -XPOST localhost:9200/baidu/baike/
_search -d'{
"query": {
     "match_all": {}
   }
}'
```

3.2.3 from/size 查询

在检索过程中，可以控制结果的规模以及从哪个结果开始返回，即在请求中可以设置相应的属性，其中：

- from：该属性指定从哪个结果开始返回(默认是 0)。
- size：该属性指定查询的结果集中包含的最大文档数(默认是 10)。

基于上述内容的查询体(不含 HTTP 方法、索引文件名、类型文件名等信息，限于篇幅，本章后面多数采取这种表示方法)如代码段 3.2 所示。

代码段 3.2：含有 from、size 属性的查询体

```
curl -H 'Content-Type: application/json' -XPOST localhost:9200/baidu/baike/
_search -d'{
  "from": 5,
  "size": 15,
  "query": {
    "term": { "content": "yoona" }
  }
}'
```

3.2.4 检索结果排序

信息检索结果集合的排序策略对用户和网站开发者双方来说都是非常重要的，因为很少有人对排名很靠后的检索结果给予充分的重视，前 10 个检索结果对所有用户来说都很重要。其实，在 Elasticsearch 的检索结果中，也可以设置不同字段对应的排序权重，然后将其应用于检索结果排序。例如，在索引文件 baidu 的类型文件 baike 下，可通过指定 fields 数组，在 title、content 两个字段中搜索给定关键字。假设设置关键字在 title 字段中出现时的权重是 2，而在 content 中出现时的权重是 1，可以参照图 3.1 中的方法(图左侧是代码实现，图右侧是针对 baike 数据集的检索结果，后同)，将 title 字段写成 title^2，即表示在查询时设置 title 字段权重为 2；而对 content 字段不加修改，其权重为默认值 1。

执行查询过程中，如果加入 sort 子句，可以不针对某个具体的字段，而是针对默认的排序分数_score 按顺序或逆序排序。代码段 3.3 给出了对 match 的查询结果的排序方法示例。

```
▶ 历史记录
▼ 查询
http://localhost:9200/baidu/baike/
_search                                    POST
{
  "query": {
    "multi_match": {
      "query": "能量",
      "fields": [
        "title^2",
        "content"
      ]
    }
  }
}
提交请求  验证 JSON  ☑ 易读
```

```
{
  "took": 11,
  "timed_out": false,
▼ "_shards": {
    "total": 5,
    "successful": 5,
    "skipped": 0,
    "failed": 0
  },
▼ "hits": {
    "total": 313,
    "max_score": 27.915726,
  ▼ "hits": [
      {
        "_index": "baidu",
        "_type": "baike",
        "_id": "y6CXpmMBMC4W8Gpsyc71",
        "_score": 27.915726,
      ▼ "_source": {
          "url": "https://baike.baidu.com/item/%E8%83%BD%E9%87
          "title": "能量",
          "content": "能量（energy）是物质的时空分布可能变化程度的度量,
          的质能关系式：E=MC²。 能量的单位与功的单位相同，在国际单位制
          单位，1电子伏=1.602,18×10－19焦。物理领域，也用尔格（erg）
          分类，能量可分为机械能、化学能、热能、电能、辐射能、核能、光能
          场也具有能量。 能量的本质是物理意义上四维空间度量的一个物理量,
          物质在不同维度所表现出来的物质属性，具体参见资料《四维空间与能
          "lastModifyTime": "2018-04-05",
```

图 3.1　设置不同字段的权重

代码段 3.3：基于_score 对结果排序

```
curl -H 'Content-Type: application/json' -XPOST localhost:9200/baidu/baike/_search -d '{
"query": {
      "match": {
            "content": "怪味豆"
      }
   },
   "sort": [{ "_score": "desc"      }]           //排序方式
}'
```

对于指定字段的排序而言，可以指定_last 将无值的结果放在检索结果的最后（如果指定为_first 则是将其放在检索结果的首位）。执行时，Elasticsearch 会开辟一部分内存空间用于保存和解析字段的倒排索引，这一功能称为 fielddata，默认禁止该功能。要实现这样的排序，需要先执行代码段 3.4 来启用 fielddata 功能，再执行代码段 3.5 完成检索和排序。

代码段 3.4：为 title 启用 fielddata 功能

```
curl -H 'Content-Type: application/json' -XPUT localhost:9200/baidu/_mapping/baike -d '{
//注意使用 PUT 方法
   "properties": {
```

```
        "title": {                                //指定的 field
          "type": "text",
          "fielddata": true                       //启用 fielddata 功能
        }
      }
    }'
```

代码段 3.5：指定默认值的排序策略

```
curl -H 'Content-Type: application/json' -XPOST localhost:9200/baidu/baike/
_search -d '{
  "query": {
    "match_all": {}
  },
  "sort": [
    {
      "title": {                                //指定的 field
        "order": "desc",
        "missing": "_last"                      //将无值的结果放在最后显示
      }
    }
  ]
}'
```

默认情况下，检索结果返回的是完整的 JSON 格式的文档。可通过_source 来引用想要返回的检索结果（即搜索结果 hits 中的_source 字段，如图 3.1 右侧所示）。也就是说，如果不想返回完整的源文档结构，可以返回指定的部分字段的检索结果集，这类似于针对关系型数据库管理系统的 SQL 查询语句中的“select 部分字段名 from 某数据表”中的“部分字段名”部分。代码段 3.6 是简化了的_source 字段，它仍会返回一个叫作_source 的字段，但是这里仅包含指定的几个字段。在图 3.2 所示的例子中，针对给定的检索词“世界历史人物”进行了分词处理，也就是说，会检索到在指定字段中包含“世界”“世界历史”“历史人物”等特征词的结果集。

代码段 3.6：通过 match 参数指定字段和检索词，排序后返回指定数量的指定字段的检索结果集

```
curl -H 'Content-Type: application/json' -XPOST localhost:9200/baidu/baike/
_search -d '{
  "query": {
    "match": {
```

```
        "taglist": "世界历史人物"
      }
    },
    "sort": {
      "lastModifyTime": {
        "order": "asc"              //按照这里指定的 lastModifyTime 字段升序排序
    }
    },
    "_source": ["url", "taglist"],  //显示的结果集
    "size": 10                      //返回的结果集数量
  }'
```

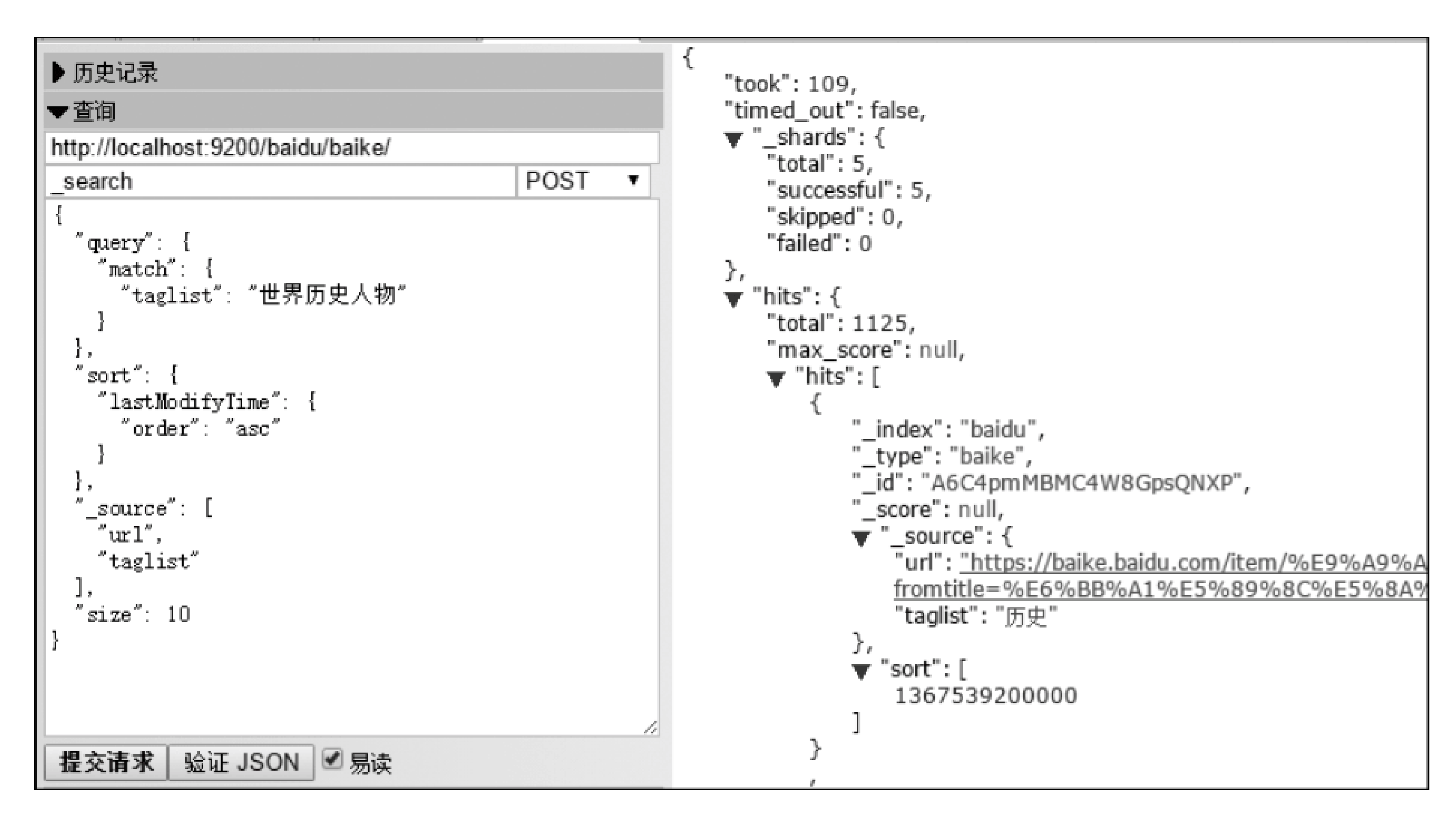


图 3.2　指定字段的检索

3.2.5　高亮搜索词

在查询中使用 highlight 子句，可以在查询结果中将每一条结果的一个或多个字段中出现的搜索词高亮显示，高亮的部分将会在查询结果后面以"highlight"：{高亮的内容}的格式显示，其中高亮的搜索词默认被标记＜strong＞＜/strong＞包围，在实际使用中，可以人为规定这一标记的写法；也可以使用 fragment_size 指定高亮显示的搜索词所在上下文的长度，以及使用 number_of_fragments 指定包含高亮搜索词的片段数量。代码段 3.7 实现了对查询结果中的搜索词“消息队列”的高亮显示，其检索结果如图 3.3 所示。

代码段 3.7：高亮显示查询结果中的搜索词

```
curl -H 'Content-Type: application/json' -XPOST localhost:9200/it-home/
posts/_search -d '{
  "query": {
    "match": {
      "content": "消息队列"                    //要高亮显示的搜索词
    }
  },
  "highlight": {
    "pre_tags": ["<strong>"],                 //规定高亮标记写法
    "post_tags": ["</strong>"],
    "fields": {
      "content": {
        "fragment_size": 10,                  //规定高亮搜索词所在上下文长度
        "number_of_fragments": 3              //规定显示多少条高亮的结果
      }
    }
  }
}'
```

```
▼ "highlight": {
  ▼ "content": [
      "如果您从工作中之听过但未有接触过<strong>消息</strong>对<strong>队列</strong>"
      ,
      "<strong>消息</strong><strong>队列</strong> 五、<strong>消息</strong><strong>队列</strong>"
    ]
}
```

图 3.3　搜索词“消息队列”被加上＜strong＞标记并高亮显示

3.2.6　查询模板

Elasticsearch 允许在查询语句中用 template 设置语句模板和参数。在执行查询时，设置好的参数将被填充到语句中对应的位置并拼成一条完整的语句执行。代码段 3.8 实现了基于模板的查询。本例将字段名、搜索词和搜索结果数量分别作为参数传入查询语句来执行查询。

代码段 3.8：基于模板进行查询

```
curl -H 'Content-Type: application/json' -XPOST localhost:9200/it-home/
posts/_search/template -d '{                 //这里注意在末尾要加上 template
  "inline": {
```

```
        "query": {
          "match": {
            "{{my_field}}": "{{my_value}}"          //设置形式参数,使用{{ }}括起来
          }
        },
      "size": "{{my_size}}"
      },
      "params": {
        "my_field": "category",                    //设置实际参数
        "my_value": "java",
        "my_size": 5
      }
    }'
```

3.3 检索进阶

Elasticsearch 提供了一个基于 JSON 格式的完整的 Query DSL,用于定义查询。Query DSL 被视为查询的抽象语法树,由简单查询、复合查询两种类型的语句组成。查询子句的作用互不相同,具体取决于它们是在查询上下文还是过滤器上下文中被使用。本节主要对 Query DSL 进行介绍,涉及全文检索、词项检索、复合查询、跨度查询、特殊查询等。

3.3.1 全文检索

全文检索通常用于在全文字段上执行查询。本节对全文检索中的 match、match_phrase、match_phrase_prefix、multi_match、simple_query_string 等查询项进行介绍。

1. match 查询

match 查询子句可接受文字、数字和日期等类型的数据,match 子句用于查询指定索引下的所有文档(如代码段 3.9 所示),也可以利用 size 指定返回结果集的大小。

```
代码段 3.9: match 子句相当于“select content from 某数据表”
curl -H 'Content-Type: application/json' -XPOST localhost:9200/it-home/
posts/_search -d '{
  "query": {
    "match": {
      "content": "程序员"
    }
  }
}'
```

代码段 3.10 演示了执行一次 match_all 查询并且返回第 11～15 个文档的代码实现。

代码段 3.10：匹配所有文档且返回 top 11～15 的记录集

```
curl -H 'Content-Type: application/json' -XPOST localhost:9200/it-home/
posts/_search -d '{
    "query": {
       "match_all": { }
    },
    "from": 10,
    "size": 5
}'
```

代码段 3.11 给出 match_all 查询并且指定字段(lastModifyTime)的升序排序的实现方法，返回满足条件的 3 条记录在关系型数据库管理系统中，相当于 SQL 语句“select top 3 * from 某数据表 order by lastModifyTime”。

代码段 3.11：匹配所有文档且检索结果按指定的字段排序，返回前 3条记录

```
curl -H 'Content-Type: application/json' -XPOST localhost:9200/baidu/baike/
_search -d '{
  "query": {
    "match_all": { }
  },
  "sort": {
    "lastModifyTime": {              //指定的排序字段
      "order": "asc"                 //排序策略
    }
  },
  "size": 3                          //返回的结果集大小
}'
```

2. match_phrase 查询

match_phrase 查询对文本进行分析，并在分析过的文本中构建一个短语查询。其中的 slop 参数定义了在查询文本的词项之间间隔多少个未知单词才视为短语匹配成功。代码实现见代码段 3.12。

代码段 3.12：match_phrase 查询

```
curl -H 'Content-Type: application/json' -XPOST localhost:9200/baidu/baike/
_search -d '{
```

```
  "query": {
    "match_phrase": {
      "title": {
        "query": "中国谚语",
        "slop": 2
      }
    }
  },
  "_source": [
    "title"
  ]
}'
```

3. match_phrase_prefix 查询

match_phrase_prefix 与 match_phrase 基本相同,不同的是这里的查询信息以短语形式出现,查询时考虑的是最后一个词的前缀匹配,其中的参数 max_expansions 表示短语中最后一个词将与多少候选词进行匹配(默认值为 50)。代码段 3.13 使用"国防"进行查询,并将 max_expansions 设置为 10,其含义是 Elasticsearch 为输入的短语中最后一个词项准备候选匹配词的数量,这里不排除候选词中未出现用户需要的词的可能性。

```
代码段 3.13: match_phrase_prefix 查询
curl -H 'Content-Type: application/json' -XPOST localhost:9200/baidu/baike/
_search -d '{
  "query": {
    "match_phrase_prefix": {
      "content": {
        "query": "国防",                    //注意要找一个存在的短语
        "max_expansions": 10
      }
    }
  }
}'
```

4. multi_match 查询

Elasticsearch 支持使用 multi_match 子句进行跨字段检索(只需写明需要检索的多个目标字段即可),并同时从多个字段中返回包含指定检索词的内容。代码段 3.14 实现了在 title、content 两个字段中对关键词"华夏"的查询。

代码段 3.14：利用 multi_match 在多个字段中检索

```
curl -H 'Content-Type: application/json' -XPOST localhost:9200/baidu/baike/
_search -d '{
  "query": {
    "multi_match": {
      "query": "华夏",
      "fields": [                    //指定在哪些字段中进行检索
        "title", "content"
      ]
    }
  }
}'
```

5. simple_query_string 查询

与其他的查询类型相比，simple_query_string 查询支持 Lucene 所有的查询语法。对于给定的内容，simple_query_string 查询使用查询解析器来构造实际的查询。示例如代码段 3.15 所示（Rafal，2015）。

代码段 3.15：simple_query_string 查询

```
curl -H 'Content-Type: application/json' -XPOST localhost:9200/baidu/baike/
_search -d '{
  "query": {
    "simple_query_string": {
      "query": "title:中国^2+ title:日本 -content:美国",
                                        //注意，这里的加号和减号前的空格不能省略
      "flags": "ALL"    //设置查询中支持的 Lucene 查询符号，如+表示 AND、-表示 NOT
    }
  }
}'
```

Tips 代码段 3.15 中出现的查询是典型的 Lucene 查询模式，其中，“title：中国^2”是指在 title 字段中要包含字符串“中国”且其权重为 2；“+title：日本”是指在 title 字段中还要同时包含字符串“日本”，只不过该字符串的权重为 1，这些权重会影响到最终结果排序；“-content：美国”是指在 content 字段中不能含有字符串“美国”。

3.3.2　词项检索

全文检索会在执行之前分析查询字符串，而词项检索是对存储在倒排索引文件中的确切词语进行操作，这些查询通常用于结构化数据(如数字、日期和枚举等)而不是全文本的内容。本节将对词项检索中的 term、terms、range、prefix、wildcard、regexp 等进行介绍。

1. term 查询

term 查询仅匹配在给定字段有某个词项的文档。例如代码段 3.16 即是 term 查询。term 查询中词项不再被解析，如果希望提升该 term 的重要性，可以在代码中增加 boost 属性。

代码段 3.16：含有 boost 的 term 查询

```
curl -H 'Content-Type: application/json' -XPOST localhost:9200/baidu/baike/
_search -d '{
  "query": {
    "term": {                    //term 查询
      "title": {                 //查询字段，给定值及 boost
        "value": "中国",
        "boost": 10
      }
    }
  }
}'
```

2. terms 查询

terms 查询允许匹配包含某些词项的文档。例如，如果想查询在 baidu/baike 文档的 title 字段中包含字符串“中国”或“日本”的文档，可以采用代码段 3.17 中的方法。

代码段 3.17：terms 查询

```
curl -H 'Content-Type: application/json' -XPOST localhost:9200/baidu/baike/
_search -d '{
  "query": {
    "terms": {
      "title": [
        "中国",
        "日本"
      ]
    }
  }
}'
```

3. range 查询

range 查询是范围查询，一般只作用在单个字段上，并且查询的参数要封装在字段名称中。它支持如下参数：

- gte：即大于或等于，表示范围下界。
- gt：即大于，表示范围下界。
- lte：即小于或等于，表示范围上界。
- lt：即小于，表示范围上界。
- boost：查询的权重，默认值为 1.0。

有关 range 查询的用法见代码段 3.18。

代码段 3.18：range 查询，其中 lastModifyTime 字段是指百度百科最近一次更新时间

```
curl -H 'Content-Type: application/json' -XPOST localhost:9200/baidu/baike/
_search -d '{
  "query": {
    "range": {                           //range 查询
      "lastModifyTime": {                //指定查询字段及其范围
        "gte": "2018-04-05",
        "lte": "2018-09-15",
        "boost": 2
      }
    }
  }
}'
```

代码段 3.19 展示了在类型文件 baike 中对日期型或时间型字段(这里是对 lastModifyTime 字段)进行范围检索的方法，它表示查找的是在从当前时间节点倒退 12h 的查询结果集，到当前时间的筛选数据子集(这里的时间可以任意加减，还可改为"now-3d"等，意为从现在开始往前 3 天起算)。

代码段 3.19：在 range 子句中设定 lastModifyTime 的上下限

```
curl -H 'Content-Type: application/json' -XPOST localhost:9200/baidu/baike/
_search -d '{
  "query": {
    "range": {
      "lastModifyTime": {
        "gt": "now-12h",
        "lt": "now"
```

```
        }
      }
    }
  }'
```

Tips 这里提到的 h 和 d 是时间单位。Elasticsearch 支持的时间单位有 y(年)、M(月)、w(周)、d(日)、h(12 小时制的时)、H(24 小时制的时)、m(分)、s(秒)。

4. prefix 查询

prefix 查询能够找到某个字段中以给定前缀开头的文档。同样，这里也支持利用 boost 属性来影响排序结果。基于 prefix 查询实现的方法见代码段 3.20。

代码段 3.20：prefix 查询

```
curl -H 'Content-Type: application/json' -XPOST localhost:9200/baidu/baike/
_search -d '{
  "query": {
    "prefix": {                       //prefix 查询
      "title": {                      //在 title 字段中查询,可指定值和权重
        "value": "中华",
        "boost": 2
      }
    }
  },
  "_source": [
    "title"
  ]
}'
```

5. wildcard 查询

wildcard(通配符)查询允许在要查询的内容中使用通配符 * 和?(通配符 * 表示任意多个任意字符，通配符?表示一个任意字符)。除此之外，它和 term 查询相似，如代码段 3.21 所示，只需把 term 查询中的 term 换为 wildcard，在查询字段中根据需要嵌入 * 或?即可。

代码段 3.21：含有 boost 的 wildcard 查询

```
curl -H 'Content-Type: application/json' -XPOST localhost:9200/baidu/baike/
_search -d '{
```

```
    "query": {
      "wildcard": {                    //wildcard 查询
        "content": {                   //查询字段,给定值及 boost
          "value": "* 豆",
          "boost": 10
        }
      }
    }
  }'
```

6. regexp 查询

在对程序员论坛数据集 it-home 的检索过程中,可能要设置相应的 from、size、boost 等属性。例如使用 regexp(正则表达式)查询的方式,在数据集中的 category 字段中搜索关键字,为了提升 category 字段的重要性,在代码中可提升其 boost 属性值。也可以同时指定多个字段的属性值。代码段 3.22 中利用 regexp 查询指定搜索结果的 category 字段必须为“Web 前端”或“数据库学习”,并获取其前 10 条结果。查询结果如图 3.4 所示。

代码段 3.22: regexp 查询

```
curl -H 'Content-Type: application/json' -XPOST localhost:9200/it-home/
posts/_search -d '{
  "from": 0,
  "size": 10,
  "query": {
    "regexp": {
      "category": {
        "value": "[Web 前端 | 数据库学习 ]",          //"Web 前端"或"数据库学习"
        "boost": 10
      }
    }
  }
}'
```

3.3.3 复合查询

复合查询语句包含了其他复合查询或简单查询,能够结合其查询结果和分值来改变其表现形式,或从查询转换为过滤器上下文。本节对复合查询中的 bool、boosting 等查询进行介绍。

图 3.4　regexp 查询

1．bool 查询

bool 查询是一种根据对结果的必要程度将不同查询子句组合起来的查询方式。在 bool 查询中可以同时使用 must、filter、should、must_not 等子句，之后可与基本检索方法里的 match 和 term 查询结合使用。代码段 3.23 是在程序员论坛数据集 it-home 中检索 category 字段中含有“开发”、java 且不含“.net”字符串的结果集，其中对 java 的查询在 filter 子句中，不考虑权重。查询结果如图 3.5 所示。

代码段 3.23：查询 category 字段中含有“开发”、java 且不含有“.net”的结果

```
curl -H 'Content-Type: application/json' -XPOST localhost:9200/it-home/
posts/_search -d '{
  "query": {
    "bool": {
      "must": {
        "match": {"category": "开发"}
      },
      "filter": {
        "term": {"category": "java"}
      },
      "must_not": {
        "term": {"category": ".net"}
```

```
            }
          }
        }
      }'
```

```
▶历史记录
▼查询
http://localhost:9200/it-home/posts/
_search/                POST ▾
{
  "query": {
    "bool": {
      "must": {
        "match": {
          "category": "开发"
        }
      },
      "filter": {
        "term": {
          "category": "java"
        }
      },
      "must_not": {
        "term": {
          "category": ".net"
        }
      }
    }
  }
提交请求  验证 JSON  ☑易读
```

```
{
  "took": 13,
  "timed_out": false,
  ▼"_shards": {
    "total": 5,
    "successful": 5,
    "skipped": 0,
    "failed": 0
  },
  ▼"hits": {
    "total": 319,
    "max_score": 1.4420223,
    ▼"hits": [
      {
        "_index": "it-home",
        "_type": "posts",
        "_id": "-aAfpmMBMC4W8GpsZr30",
        "_score": 1.4420223,
        ▼"_source": {
          "url": "http://bbs.it-home.org/thread-105131-1-1.html",
          "category": "Java开发",
          "title": "[Java学习] Java类和对象的概念",
          "user": "半夏花开",
          "publishTime": "4 天前",
          "content": "Java是一门面向对象的编程语言，理解Java，首先要理解类与对象这两个概念
          据类型，可以包含不同的成员（变量），每个成员的数据类型可以不一样；可以通过结构体来
          <stdio.h> 2. int main(){ 3.    // 定义结构体 Student 4.    struct Student{ 5.
            float score; 9.    }; 10. 11.    // 通过结构体来定义变量 12.    struct Student stu
```

图 3.5　bool 查询的结果

2. boosting 查询

boosting 查询可将匹配到的结果降级处理。与带有 not 子句的 bool 查询不同，boosting 查询结果中仍然会包含不符合预期的词项，但其分值会降低。代码段 3.24 实现了在有关 java 的分类中查询内容与 json 不相关的信息。关键词 json 处于 negative 子句中，其结果的分值将会根据 negative_boost 的值被相应降低。

代码段 3.24: boosting 查询

```
curl -H 'Content-Type: application/json' -XPOST localhost:9200/it-home/
posts/_search -d '{
  "query": {
    "boosting": {
      "positive": {                     //预期相关的信息
        "term": {
          "category": "java"
        }
      },
      "negative": {                     //预期不相关的信息
        "term": {
```

```
            "content": "json"
          }
        },
        "negative_boost": 0.2                    //不相关信息要降低的分值
      }
    }
  }'
```

3.3.4 跨度查询

跨度查询是一种低级别的、基于词项相对位置的查询，可以对指定项的顺序和接近度进行控制，可用于对法律文件或专利等的非常具体的查询。除 span_multi 查询之外，跨度查询不能与非跨度查询复合使用。本节对 span_term、span_or、span_containing、span_within 等查询进行介绍。

1. span_term 查询

span_term 查询可以查询含有查询词项的一段文本，这一功能相当于 Lucene 中的 SpanTermQuery。代码段 3.25 实现了对含有词项 java 的文本的查询。

代码段 3.25：span_term 查询

```
curl -H 'Content-Type: application/json' -XPOST localhost:9200/it-home/
posts/_search -d '{
  "query": {
    "span_term": {
      "content": {
        "value": "java",
        "boost": 2
      }
    }
  }
}'
```

2. span_or 查询

span_or 查询可以匹配 span_term 子句查询结果的并集，这一功能相当于 Lucene 中的 SpanOrQuery。代码段 3.26 实现了对含有词项 java、json 或 jquery 的文本的查询。

代码段 3.26：查询含有词项 java、json 或 jquery 的文本

```
curl -H 'Content-Type: application/json' -XPOST localhost:9200/it-home/
posts/_search -d '{
```

```
    "query": {
      "span_or": {
        "clauses": [
          {"span_term": {"content": "java"}},
          {"span_term": {"content": "json"}},
          {"span_term": {"content": "jquery"}}
        ]
      }
    }
  }'
```

3. span_containing 查询

span_containing 查询含有 little 子句和 big 子句，在查询时一旦发现 big 子句的匹配项中包含了 little 子句中的匹配项，那么 big 子句的匹配项将作为查询结果返回。该查询功能相当于 Lucene 中的 SpanContainingQuery。代码段 3.27 实现了在词项“出现”和“错误”的匹配项中对含有词项 500 的匹配项的查询，结果如图 3.6 所示，图中的方框标出了词项“出现”和“错误”在文中形成的跨度，而词项 500 正处于其中。其中 big 子句中的 span_near 查询是一种查找邻近词项的跨度查询，后面的 slop 指定了两个词项之间可以存在多少个不匹配的词项。

代码段 3.27：查询词项“出现”和“错误”的匹配项中含有词项 500 的匹配项的结果

```
curl -H 'Content-Type: application/json' -XPOST localhost:9200/it-home/
posts/_search -d '{
  "query": {
    "span_containing": {
      "little": {
        "span_term": {"content": "500"}              //跨度中的词项
      },
      "big": {
        "span_near": {
          "clauses": [
            {"span_term": {"content": "出现"}},       //两个词项确定了一个跨度
            {"span_term": {"content": "错误"}}
          ],
          "slop": 10,
          "in_order": true
        }
```

```
            }
        }
    }
}'
```


"_score": 7.197392,
▼ "_source": {
    "url": "http://bbs.it-home.org/thread-99871-1-1.html",
    "category": "PHP编程",
    "title": "[PHP学习] ubuntu nginx php7.1",
    "user": "小贝",
    "publishTime": "2018-1-26 08:00:02",
    "content": "sudo apt-get update sudo apt-get install -y language-pack-en-base locale-gen en_US.UTF-8 sudo apt-get install software-properties-common sudo LC_ALL=en_US.UTF-8 add-apt-repository ppa:ondrej/php sudo apt-get update sudo apt-get -y install php7.1 sudo apt-get -y install php7.1-mysql sudo apt-get install php7.1-fpm apt-get install php7.1-curl php7.1-xml php7.1-mcrypt php7.1-json php7.1-gd php7.1-mbstring sudo apt-get -y install nginx sudo apt-get -y install mysql-server-5.6 原文链接：https://www.cnblogs.com/php-linux/p/6016439.html nginx 配置文件： server { listen 80; listen [::]:80; server_name abc.cn; root /var/www/abc_cn; index index.html index.php; location / { try_files $uri $uri/ /index.php?$query_string; } location ~ \.php$ { include snippets/fastcgi-php.conf; fastcgi_pass unix:/run/php/php7.1-fpm.sock; } } 因为复制之前的配置文件，使用的是fastcgi_pass unix:/run/php/php7.0-fpm.sock;，导致出现502错误，和访问php文件直接下载了，把7.0换成7.1后，就解决了。 如果出现500错误，就是权限问题 chmod -R 777 ."
  }
}

图 3.6　查询两个词项所确定的跨度中包含的特定词项匹配项

4. span_within 查询

span_within 查询也含有 little 子句和 big 子句，在查询时一旦发现 little 子句的匹配项包含在 big 子句中的匹配项中，那么 little 子句的匹配项将作为查询结果返回，该查询功能相当于 Lucene 中的 SpanWithinQuery。代码段 3.28 实现了对词项“出现”和“错误”所确定的跨度中含有词项 500 的匹配项的查询，查询结果同图 3.6。

代码段 3.28：查询词项“出现”和“错误”所确定的跨度中含有词项 500 的匹配项

```
curl -H 'Content-Type: application/json' -XPOST localhost:9200/it-home/
posts/_search -d '{
  "query": {
    "span_within": {
      "little": {
        "span_term": {"content": "500"}              //跨度中的词项
      },
      "big": {
        "span_near": {
          "clauses": [
            {"span_term": {"content": "出现"}},     //两个词项确定了一个跨度
            {"span_term": {"content": "错误"}}
          ],
          "slop": 10,
```

```
                "in_order": true
              }
            }
          }
        }
      }'
```

3.3.5 特殊查询

除上面提到的几类查询方式以外，还有一些特殊的查询。本节对 more_like_this 查询进行介绍，3.3.6 节对 script 查询进行介绍。more_like_this 查询得到与指定文本相似的文档，在这里可以使用的部分参数如下（Rafal，2015）：

- fields：查询所作用的字段的列表，默认是_all。
- like_text：指明文档应比较的内容。
- min_term_freq：指定文档中词项出现的最低频次，低于该值的词项将被忽略，默认是 2。
- min_doc_freq：指定词项应至少在多少个文档中出现才不会被忽略，默认是 5。
- min_word_length：指定单个单词的最小长度，低于该值的单词将被忽略，默认是 0。
- max_query_terms：指定在生成的查询中查询词项的最大数目，默认是 25。
- max_doc_freq：指定含有被搜索词项的文档出现的最高频次，高于该值时，将忽略输入文档中的词项，默认是 0，即无限制。
- max_word_length：指定单个单词的最大长度，高于该值的词项将被忽略，默认是 0，即无限制。
- stop_words：指定忽略词集。
- boost：提升一个查询的权重时使用的权重，默认是 1.0。
- analyzer：指定用于分析内容的分词器。

more_like_this 查询示例代码如代码段 3.29 所示，该代码段查询与指定的两条存储在 Elasticsearch 中的信息以及一条额外的信息相似的结果。其中，在 fields 子句中指定了 title 和 content 两个字段，这是 more_like_this 查询的执行范围；在 like 数组中前两个对象分别指定了 it-home 索引中 posts 类型下的某个特定 id 中的文档内容，而后面的文本是额外的自定义查询字符串；min_term_freq 表示信息在文档中出现的最小次数；max_query_terms 表示要显示的匹配结果的最大条数。

```
代码段 3.29: more_like_this 查询
curl -H 'Content-Type: application/json' -XPOST localhost:9200/it-home/
posts/_search -d '{
  "query": {
    "more_like_this": {                     //more_like_this 查询
      "fields": ["title", "content"],       //查询字段
      "like": [                             //指定应比较的内容
        {
          "_index": "it-home",
          "_type": "posts",                 //对齐
          "_id": "AViVbrJPOQdYmPY46b3B"
        },
        {
          "_index": "it-home",
          "_type": "posts",
          "_id": "AViVb4p20QdYmPY46b5T"
        },
        "现在我们来写一个测试程序"           //可以直接指定额外的相关内容
      ],
      "min_term_freq": 1,                   //表示信息在文档中最少出现多少次才能被检出
      "max_query_terms": 12                 //规定显示结果的最大条数
    }
  }
}'
```

3.3.6　脚本

脚本(script)模块可以使用 script 子句来编辑表达式，当执行查询时，script 子句可以生成一个由外部参数计算出的特定字段，插入到原来的查询语句中执行；或者在查询中通过自定义的计算方法算出搜索结果的得分。script 子句默认使用的脚本语言是 painless。一般情况下子句中包含 3 条语句，下面给出 script 子句的一般格式：

```
"script": {
    "lang":   "...",                              //这里设置使用何种脚本语言
    "inline" | "stored" | "file": "...",          //这里编辑表达式，给出形式参数
    "params": { ... }                             //这里给出实际参数
}
```

Elasticsearch 6.0 以上的版本默认开启 script 查询功能。但是为了提高安全性，不要以

root 用户启动，不要将 Elasticsearch 操作直接暴露给用户，不要将 Elasticsearch 直接暴露在互联网上。另外，Elasticsearch 提供了两个配置项，可以设置 script 查询的类型和内容。例如，在 Elasticsearch 的配置文件 elasticsearch.yml 中添加以下两行配置信息，允许脚本搜索和更新，允许脚本类型为 inline：

```
script.allowed_contexts: search, update        //默认为 none,允许所有执行内容
script.allowed_types: inline                   //默认为 none 允许所有类型
```

设置完成后重新启动 Elasticsearch。

> Tips　Groovy、JavaScript 和 Python 在 Elasticsearch 5.0 时已被官方弃用，在 Elasticsearch 6.0 中已被完全移除。

script 查询允许使用脚本为查询提供数据。在查询中定义参数并填充到查询语句中，形成一条完整的查询语句。script 查询可以被编译和缓存以更快地执行。如果在编写查询语句时，多段代码除少量参数不同之外基本相同，最好使用 script 语句，只要将不同参数分别传递到语句中即可。代码段 3.30 实现了在 whale 索引文件下的 log 类型文件中的查询，脚本查询条件为日志的 size 大于 160，结果如图 3.7 所示。这里需要注意，字段 log_size 的数据类型必须设置为 long，否则类型不匹配，查询将失败。

代码段 3.30：查询 whale 索引文件中日志的 size 大于 160 的记录

```
curl -H 'Content-Type: application/json' -XPOST localhost:9200/whale/log/
_search -d '{
  "query": {
    "bool": {
      "must": {
        "script": {
          "script": {
            "inline": "doc['log_size'].value>params.size",     //设置形式参数
            "lang": "painless",
            "params": {"size": 160}                            //给出实际参数
          }
        }
      }
    }
  }
}'
```

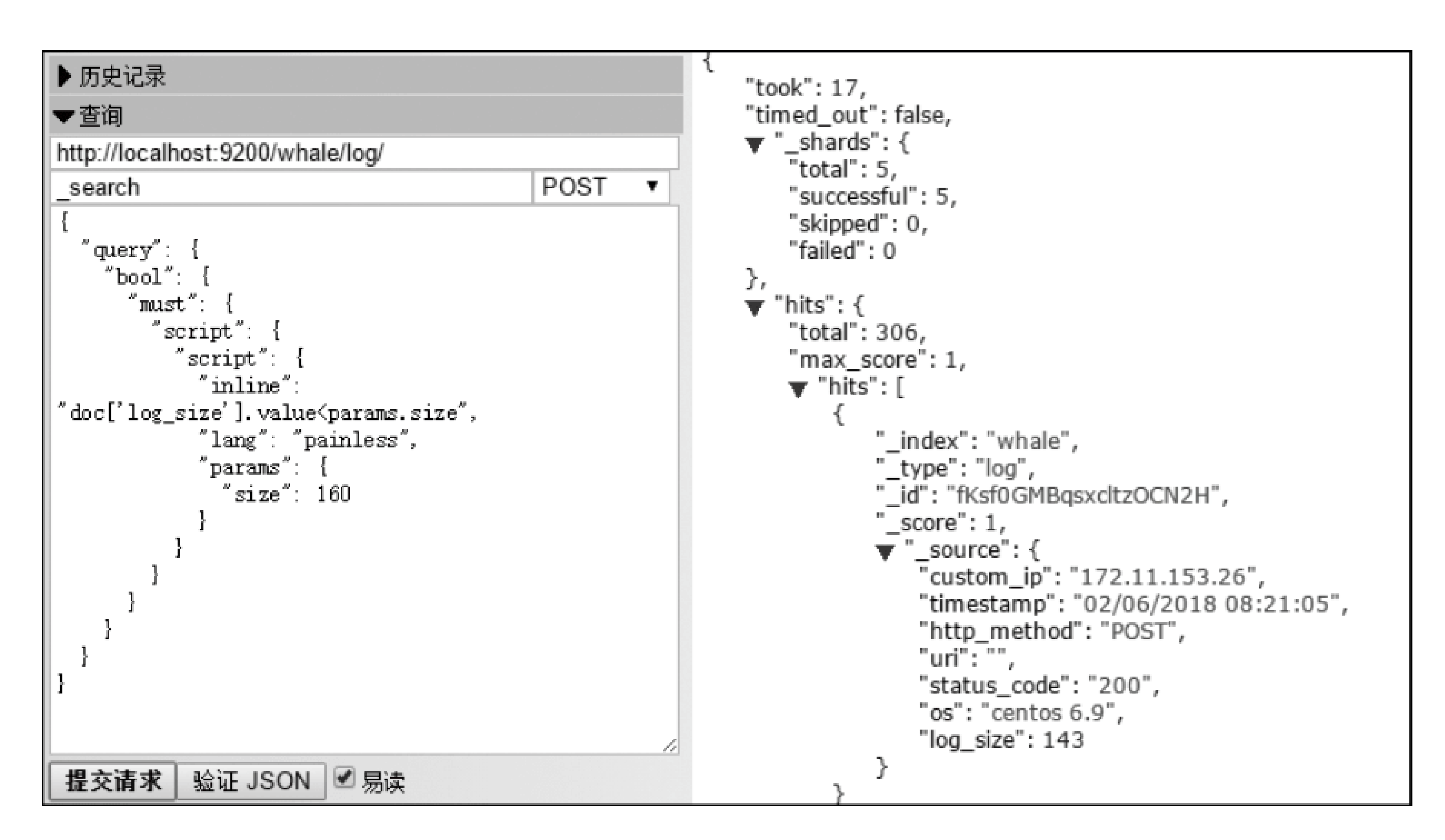


图 3.7　查询 whale 索引文件中日志的 size 大于 160 的记录的结果

在执行更新操作时也可以使用 script 查询，可以将参数传递给 inline 子句中的 ctx._source.{字段名}以进行数据更新。代码段 3.31 实现了对某条记录作者的修改。其中，inline 子句中的 ctx._source 是指访问文档中的_source 字段，_source 字段是文档中所有字段的集合，例如代码段 3.6、3.12 和 3.20 等处均指定了显示_source 字段中的一部分；lang 语句指定了脚本中使用 painless 语言；params 子句用于向 inline 子句中的形式参数 params.name 传递实际参数。

代码段 3.31：使用 script 修改某条记录的作者

```
curl -H 'Content-Type: application/json' -XPOST localhost:9200/it-home/
posts/AViVbpg-OQdYmPY46b20/_update -d '{ //指定了 id号
  "script": {
    "inline": "ctx._source.user=params.name",
    "lang": "painless",
    "params": {"name": "test"}
  }
}'
```

另外，script 查询还支持在 inline 子句中使用类似编程语言的方法编写表达式。代码段 3.32 将doc['log_size']作为变量使用，实现了对服务器日志记录中日志长度的计算。在 script_fields 子句中，total_size 是其设置的名称。在 inline 子句中给出了计算日志总长的代码。

代码段 3.32：使用 script 查询计算日志长度

```
curl -H 'Content-Type: application/json' -XPOST localhost:9200/whale/log/
_search -d '{
  "query": {
    "match_all": {}
  },
  "script_fields": {
    "total_size": {
      "script": {
        "lang": "painless",
        "inline": "int total=0; for (int i=0; i<doc['log_size'].length;++i)
          { total+=doc['log_size'][i]; }return total;"
      }
    }
  }
}'
```

3.4 聚合

聚合(aggregation)可以看作对查询结果的汇总。例如,先查询出一天内各个时间段的HTTP 请求,然后统计每天的数据。聚合真正强大的功能在于它能嵌套并实现多级汇总,每一种聚合都有自己的目的和输出,它们通常分为 4 类：metric、bucket、pipeline 和 matrix。本节对这 4 类聚合进行介绍。

metric 是对某个 bucket 中的文档计算得到的统计信息,它和关系型数据库使用的 SQL 语句中的集函数(如 count()、max()等)的作用相似。可见,聚合是由一个或多个 bucket、零个或多个 metric 组合而成的统计结果。每个文档中的值会被计算,以决定它们是否匹配了某些 bucket 的条件,如果匹配成功,那么该文档会被置入相应的 bucket 中。一个 bucket 也能够嵌套在其他的 bucket 中(即 bucket 是可以嵌套的)。对 metric 而言,多数仅使用文档中的值进行简单计算。另外,聚合也支持排序等属性,这将在相关例子中进行说明。

bucket 是满足某个条件的文档集合,它和关系型数据库使用的 SQL 语句中的 group by 子句的作用相似(但又不一样)。例如,在校大学生要么属于本科生群的 bucket,要么属于研究生群的 bucket。

pipeline 是将其他聚合的输出及其关联度量进行聚合的方式。

matrix 在多字段上操作,从请求的文档字段中提取信息,返回矩阵结果。

3.4.1　metric 聚合

metric 聚合基于从文档中提取的值来计算指标。这些值通常使用字段数据，也可以使用脚本生成。本节对 metric 聚合中的 min、max、sum、avg、stats、extended_stats、value_count 等聚合进行介绍。

1. min、max、sum、avg 聚合

min、max、sum、avg 聚合可以方便地完成对最值、和、平均值等的统计。代码段 3.33 完成对指定字段的最小值聚合，结果如图 3.8 所示（当统计最大值时，将代码段 3.33 中的统计函数换成 max 即可，这里不再赘述）。

代码段 3.33：执行最小值聚合

```
curl -H 'Content-Type: application/json' -XPOST localhost:9200/whale/log/
_search -d '{
  "aggs": {
    "min_size": {              //聚合的名称
      "min": {                 //统计最小值。当统计最大值时，将 min 换成 max 即可
        "field": "log_size"    //统计字段
      }
    }
  }
}'
```

```
▼查询
http://localhost:9200/whale/log/
_search                POST
{
  "aggs": {
    "min_size": {
      "min": {
        "field": "log_size"
      }
    }
  }
}
```

```
"took": 7,
"timed_out": false,
"_shards": { "total": 5,"successful": 5,"
"hits": {
  "total": 1001,
  "max_score": 1,
  "hits": [ { "_index": "whale","_type
},
"aggregations": {
  "min_size": {
    "value": 51
  }
}
}
```

图 3.8　最小值聚合结果

类似地，对和、平均值的统计只需在代码的相应位置使用 sum、avg 即可。代码段 3.34 完成对指定字段的平均值聚合，而针对 size 字段（HTTP 响应大小）的结果如图 3.9 所示。

代码段 3.34：执行平均值聚合

```
curl -H 'Content-Type: application/json' -XPOST localhost:9200/whale/log/
_search -d '{
  "aggs": {
    "avg_size": {                    //聚合的名称
      "avg": {                       //统计平均值。当求和时,将 avg 换成 sum 即可
        "field": "log_size"
      }
    }
  }
}'
```

```
▶ 历史记录
▼ 查询
http://localhost:9200/whale/log/
_search                POST ▼
{
  "aggs": {
    "avg_size": {
      "avg": {
        "field": "log_size"
      }
    }
  }
}
```

```
{
  "took": 11,
  "timed_out": false,
  ▼ "_shards": {
    "total": 5,
    "successful": 5,
    "skipped": 0,
    "failed": 0
  },
  ▼ "hits": {
    "total": 1001,
    "max_score": 1,
    ► "hits": [ { "_index": "whale","_type": "log","_id": "M
  },
  ▼ "aggregations": {
    ▼ "avg_size": {
      "value": 276.64835164835165
    }
  }
}
```

图 3.9　平均值聚合结果

在完成上述计算时,同样支持 script 语句的使用。代码段 3.35 中使用 script 语句计算 log_size 的最大值,结果如图 3.10 所示。

代码段 3.35：执行平均值聚合

```
curl -H 'Content-Type: application/json' -XPOST localhost:9200/whale/log/
_search -d '{
  "aggs": {
    "max_with_script": {
      "max": {
        "script": {
          "inline": "doc['log_size'].value",     //在 inline 子句中指定 log_size
          "lang": "painless"
        }
```

```
            }
        }
    }
}'
```

```
▶历史记录
▼查询
http://localhost:9200/whale/log/
_search                POST ▼
{
  "aggs": {
    "max_with_script": {
      "max": {
        "script": {
          "inline": "doc['log_size'].value",
          "lang": "painless"
        }
      }
    }
  }
}
```

```
{
  "took": 12,
  "timed_out": false,
  ▼ "_shards": {
    "total": 5,
    "successful": 5,
    "skipped": 0,
    "failed": 0
  },
  ▼ "hits": {
    "total": 1001,
    "max_score": 1,
    ► "hits": [ { "_index": "whal
  },
  ▼ "aggregations": {
    ▼ "max_with_script": {
      "value": 500
    }
  }
}
```

图 3.10 使用 script 语句执行最大值聚合

2. stats、extended_stats 聚合

stats 聚合完成多值统计，返回值包括计数、最小值、最大值、平均值、和等。代码段 3.36 演示了对相应字段进行多值统计的方法，针对类型文件 log 的返回结果如图 3.11 所示。同样，stats 聚合也支持 script 语句和参数。

代码段 3.36：执行 stats 聚合

```
curl -H 'Content-Type: application/json' -XPOST localhost:9200/whale/log/
_search -d '{
    "aggs": {
        "log_size_stats": {
            "stats": {
              "field": "log_size"
            }
        }
    }
}'
```

extended_stats 聚合则是对 stats 聚合的功能扩展，可以在上述输出结果的基础上添加

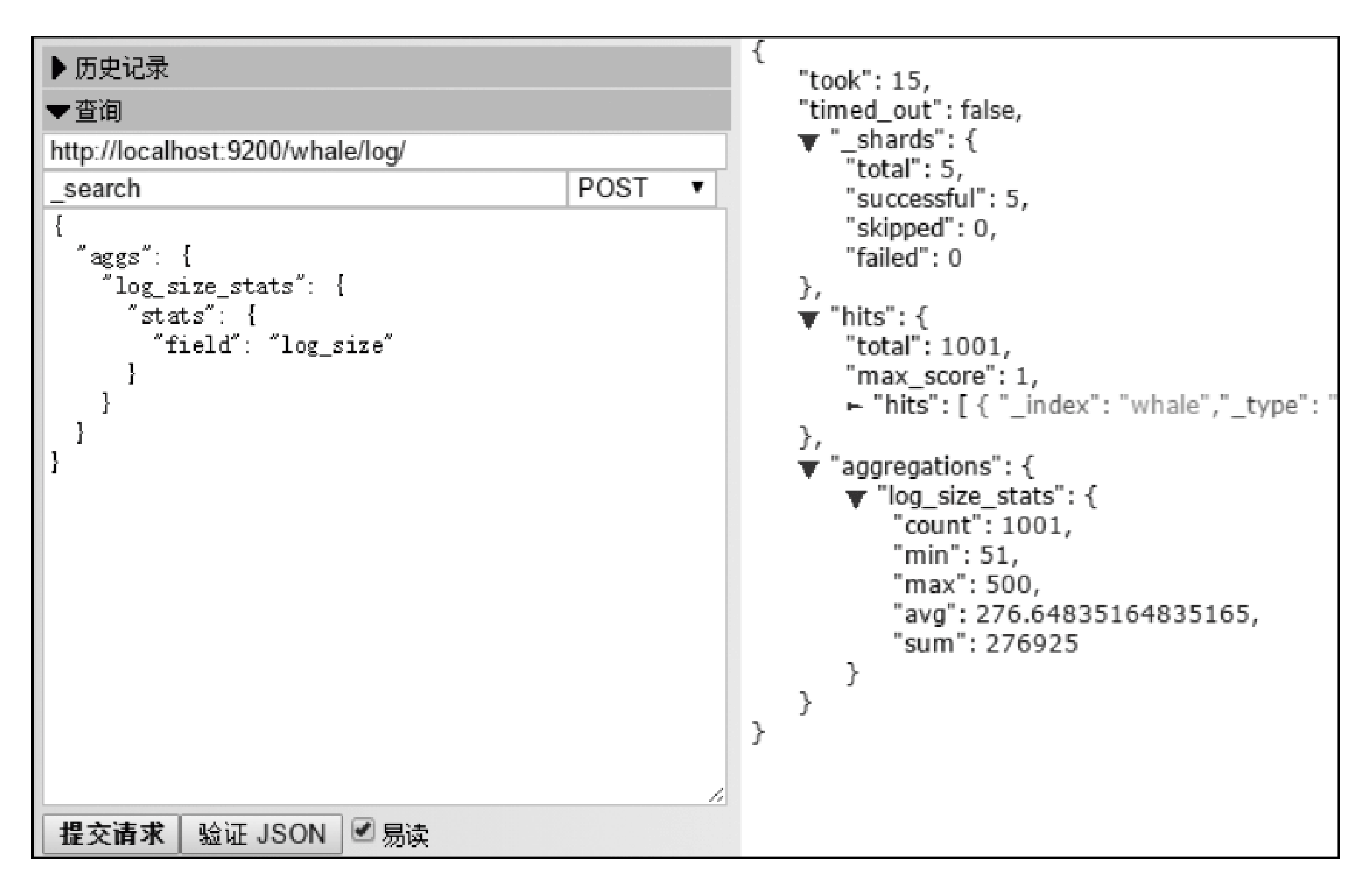


图 3.11　stats 聚合结果

平方和、方差和标准差等指标，参见代码段 3.37，运行结果如图 3.12 所示。同样地，extended stats 聚合也支持 script 语句和参数。

代码段 3.37：执行 extended_stats 聚合

```
curl -H 'Content-Type: application/json' -XPOST localhost:9200/whale/log/
_search -d '{
  "aggs": {
    "log_size_extended_stats": {
      "extended_stats": {
        "field": "log_size"
      }
    }
  }
}'
```

3. value_count 聚合

value_count 聚合是一种单值指标聚合，用来计算文档中某个字段的统计数据。首先执行代码段 3.38，为 text 类型的 custom_ip 字段开启 fielddata 功能；然后执行代码段 3.39，对该字段执行 value_count 聚合，统计某服务器日志记录的访问次数。聚合结果如图 3.13 所示。

▶ 历史记录
▼ 查询
http://localhost:9200/whale/log/
_search　POST

```
{
  "aggs": {
    "log_size_extended_stats": {
      "extended_stats": {
        "field": "log_size"
      }
    }
  }
}
```

提交请求　验证 JSON　易读
▶ 结果转换器
▶ 重复请求
▶ 显示选项

```
{
  "took": 11,
  "timed_out": false,
  "_shards": {
    "total": 5,
    "successful": 5,
    "skipped": 0,
    "failed": 0
  },
  "hits": {
    "total": 1001,
    "max_score": 1,
    "hits": [ { "_index": "whale","_type": "log","_i
  },
  "aggregations": {
    "log_size_extended_stats": {
      "count": 1001,
      "min": 51,
      "max": 500,
      "avg": 276.64835164835165,
      "sum": 276925,
      "sum_of_squares": 93733215,
      "variance": 17105.264954825398,
      "std_deviation": 130.78709781482803,
      "std_deviation_bounds": {
        "upper": 538.2225472780077,
        "lower": 15.07415601869559
      }
    }
  }
}
```

图 3.12　extended_stats 聚合结果

代码段 3.38：为字段 custom_ip 开启 fielddata 功能

```
curl -H 'Content-Type: application/json' -XPUT localhost:9200/whale/_mapping/log -d '{
                                                  //这里使用 PUT 方法
  "properties": {
    "custom_ip": {                                //要开启 fielddata 功能的字段
      "type": "text",                             //字段的类型,应与实际类型一致
      "fielddata": true
    }
  }
}'
```

代码段 3.39：执行 value_count 聚合，统计访问次数

```
curl -H 'Content-Type: application/json' -XPOST localhost:9200/whale/log/
_search -d '{
  "aggs": {
    "custom_ip_count": {
      "value_count": {
        "field": "custom_ip"
```

```
            }
        }
    }
}'
```

```
▶ 历史记录
▼ 查询
http://localhost:9200/whale/log/
_search                          POST ▼
{
  "aggs": {
    "custom_ip_count": {
      "value_count": {
        "field": "custom_ip"
      }
    }
  }
}
```

```
{
  "took": 169,
  "timed_out": false,
▼ "_shards": {
    "total": 5,
    "successful": 5,
    "skipped": 0,
    "failed": 0
  },
▼ "hits": {
    "total": 1001,
    "max_score": 1,
  ▸ "hits": [ { "_index": "whale","_t
  },
▼ "aggregations": {
  ▼ "custom_ip_count": {
      "value": 1001
    }
  }
}
```

图 3.13　value_count 聚合结果

3.4.2　bucket 聚合

本节对 terms、range、date_range、histogram、date_histogram、filter 聚合进行介绍。

1. terms 聚合

terms 聚合用于对指定字段的内容进行分布统计。聚合过程中会动态构建多个 bucket，并对每个 bucket 计算出一个特定值。代码段 3.40 实现了对某服务器上的用户请求 HTTP 服务状态码的统计，结果如图 3.14 所示。注意，代码中的 status_code 字段也需要开启 fielddata，方法参照代码段 3.38。

代码段 3.40：执行 terms 聚合，统计用户请求 HTTP 服务状态码

```
curl -H 'Content-Type: application/json' -XPOST localhost:9200/whale/log/
_search -d '{
  "aggs": {
    "usage": {
      "terms": {
        "field": "status_code"
      }
    }
  }
}'
```

```
▼ "aggregations": {
  ▼ "usage": {
      "doc_count_error_upper_bound": 0,
      "sum_other_doc_count": 0,
    ▼ "buckets": [
        {
          "key": "200",
          "doc_count": 843
        }
        ,
        {
          "key": "503",
          "doc_count": 59
        }
        ,
        {
          "key": "302",
          "doc_count": 52
        }
        ,
        {
          "key": "404",
          "doc_count": 47
        }
      ]
    }
}
```

图 3.14 terms 聚合

2. range 聚合

range 聚合基于多个 bucket，在每个 bucket 中定义一个范围，用于统计指定字段在该范围的值。在聚合过程中，利用从每个文档中提取的值针对指定的范围进行检查，并且返回相关或匹配的文档。代码段 3.41 实现了对 log_size 字段分别在 3 个范围内的数量统计，结果如图 3.15 所示。

代码段 3.41：执行 3个不同范围的 range 聚合

```
curl -H 'Content-Type: application/json' -XPOST localhost:9200/whale/log/
_search -d '{
  "aggs": {
    "log_size_ranges": {
      "range": {
        "field": "log_size",
        "ranges": [
          {"to": 300},
          {"from": 200, "to": 500},
          {"from": 200}
        ]
      }
    }
  }
}'
```

```
▼ "aggregations": {
  ▼ "log_size_ranges": {
    ▼ "buckets": [
        {
            "key": "*-300.0",
            "to": 300,
            "doc_count": 548
        }
        ,
        {
            "key": "200.0-500.0",
            "from": 200,
            "to": 500,
            "doc_count": 654
        }
        ,
        {
            "key": "200.0-*",
            "from": 200,
            "doc_count": 657
        }
      ]
  }
```

图 3.15　range 聚合结果

在 range 聚合中可以加入 keyed 参数并设置为 true，这样可以将返回值中的 key 作为这个 JSON 对象的名称。也可以使用 key 参数自定义 key 的名称。代码段 3.42 针对类型文件 log 分析字段 log_size 在 3 个不同范围内的统计值，结果如图 3.16 所示。同样，range 聚合也支持 script 语句。

代码段 3.42：使用 keyed 参数，执行 3 个不同范围的 range 聚合

```
curl -H 'Content-Type: application/json' -XPOST localhost:9200/whale/log/
_search -d '{
  "aggs": {
    "log_size_ranges": {
      "range": {
        "field": "log_size",
        "keyed": true,
        "ranges": [
          {"key": "short", "to": 200},
          {"key": "middle", "from": 200, "to": 300},
          {"key": "long", "from": 300}
        ]
      }
    }
  }
}'
```

```
▼ "aggregations": {
  ▼ "log_size_ranges": {
    ▼ "buckets": {
      ▼ "short": {
          "to": 200,
          "doc_count": 344
        },
      ▼ "middle": {
          "from": 200,
          "to": 300,
          "doc_count": 204
        },
      ▼ "long": {
          "from": 300,
          "doc_count": 453
        }
      }
    }
}
```

图 3.16 range 聚合结果

3. date_range 聚合

date_range 聚合是专门对时间类型的字段进行区段统计的聚合。代码段 3.43 介绍了其基本使用方法,在特定时间区段中日志数量的统计结果如图 3.17 所示。

代码段 3.43:执行特定时间区段的聚合

```
curl -H 'Content-Type: application/json' -XPOST localhost:9200/whale/log/
_search -d '{
  "aggs": {
    "range": {
      "date_range": {
        "field": "timestamp",
        "format": "dd/MM/yyyy HH:mm:ss",        //根据 Elasticsearch 中的实际数
                                                //据设置时间格式
        "ranges": [
          {"from": "29/05/2018 10:00:00",        //起始时间
           "to": "29/05/2018 12:00:00"}          //截止时间
        ]
      }
    }
  }
}'
```

4. histogram 聚合

histogram 聚合可以根据返回值(针对数值型或日期型的字段)生成将来可创建柱状图的聚合数据。代码段 3.44 计算 log_size 字段以 200 个字符为步长划分的各区段的统计分

```
▼ "aggregations": {
  ▼ "range": {
    ▼ "buckets": [
        {
          "key": "29/05/2018 10:00:00-29/05/2018 12:00:00",
          "from": 1527588000000,
          "from_as_string": "29/05/2018 10:00:00",
          "to": 1527595200000,
          "to_as_string": "29/05/2018 12:00:00",
          "doc_count": 9
        }
      ]
    }
}
```


图 3.17　指定时间区段的日志数量统计结果

布情况，针对类型文件 log 的返回结果如图 3.18 所示。通过聚合，可以得到 log_size 在各个区段的数量。在这个基础上，可以利用可视化软件将上述数据制作成柱状图。

代码段 3.44：执行柱状图聚合，统计 log_size 字段在固定间隔不同区段中的数量

```
curl -H 'Content-Type: application/json' -XPOST localhost:9200/whale/log/
_search -d '{
  "aggs": {
    "log_size_histogram": {
      "histogram": {
        "field": "log_size",
        "interval": 200,                    //步长为 200
        "order": {
          "_key": "desc"                    //降序排序
        }
      }
    }
  }
}'
```


```
▼ "aggregations": {
  ▼ "log_size_histogram": {
    ▼ "buckets": [
        {
          "key": 400,
          "doc_count": 231
        }
        ,
        {
          "key": 200,
          "doc_count": 426
        }
        ,
        {
          "key": 0,
          "doc_count": 344
        }
      ]
    }
}
```


图 3.18　histogram 聚合结果

也可以在 histogram 聚合中添加子聚合，即在现有的聚合中嵌套统计。代码段 3.45 实现了对每一个聚合中再次进行 stats 聚合并按照子聚合中的最小值字段（即代码中的 size_stats.min）进行降序排序，针对类型文件 log 的运行结果如图 3.19 所示。

代码段 3.45：执行 histogram 聚合，统计 log_size 字段在固定间隔不同区段中的数量

```
curl -H 'Content-Type: application/json' -XPOST localhost:9200/whale/log/
_search -d '{
  "aggs": {
    "outer_bucket": {
      "histogram": {
        "field": "log_size",
        "interval": 500,
        "order": {"size_stats.min": "asc"}
      },
      "aggs": {
        "size_stats": {
          "stats": {"field": "log_size"}
        }
      }
    }
  }
}'
```

在上述例子中可以为中间结果取名，只需使用 keyed 参数并将其设置为 true，即在代码段 3.45 的 field 和 interval 语句中间加入语句“"keyed"：true”并在末尾加逗号。这样，前端程序可以通过这个名字取得相应的统计结果并进行展示，该聚合的查询结果如图 3.20 所示。请注意图 3.19 和图 3.20 的区别：图 3.19 的 buckets 是以数组方式给出的，而图 3.20 的 buckets 则是以键值对的形式给出的。

5. date_histogram 聚合

date_histogram 聚合是一个增强型的专门用于日期型字段统计的 histogram 聚合，它允许使用 year、month、week、day、hour、minute 等常量作为 interval 属性的取值。代码段 3.46 实现了以下聚合操作：在 field 中填写一个日期类型的字段名称，在 interval 中填写一个步长值，通过 format 参数设置时间格式，结果如图 3.21 所示。

```
▼ "aggregations": {
  ▼ "outer_bucket": {
    ▼ "buckets": [
        {
          "key": 0,
          "doc_count": 998,
          ▼ "size_stats": {
              "count": 998,
              "min": 51,
              "max": 499,
              "avg": 275.97695390781564,
              "sum": 275425
          }
        }
        ,
        {
          "key": 500,
          "doc_count": 3,
          ▼ "size_stats": {
              "count": 3,
              "min": 500,
              "max": 500,
              "avg": 500,
              "sum": 1500
          }
        }
      ]
  }
}
```

图 3.19 执行嵌套聚合的结果

```
▼ "aggregations": {
  ▼ "outer_bucket": {
    ▼ "buckets": {
      ▼ "0.0": {
          "key": 0,
          "doc_count": 998,
          ▼ "size_stats": {
              "count": 998,
              "min": 51,
              "max": 499,
              "avg": 275.97695390781564,
              "sum": 275425
          }
        },
      ▼ "500.0": {
          "key": 500,
          "doc_count": 3,
          ▼ "size_stats": {
              "count": 3,
              "min": 500,
              "max": 500,
              "avg": 500,
              "sum": 1500
          }
        }
      }
    }
}
```

图 3.20 为中间结果取名的 histogram 聚合

代码段 3.46：执行 date_histogram 聚合，统计日志记录在不同时间段中的数量

```
curl -H 'Content-Type: application/json' -XPOST localhost:9200/whale/log/
_search -d '{
  "aggs": {
    "logs_over_time": {
        "date_histogram": {
        "field": "timestamp",
        "interval": "3h",              //步长为 3h
        "format": "dd/MM/yyyy HH:mm:ss"          //根据 Elasticsearch 中的实际数据设置时间格式
      }
    }
  }
}'
```

也可以在其中添加子聚合（即聚合嵌套）来实现更丰富的统计功能。代码段 3.47 实现了以 3h 为步长，统计在每小时各个状态码的出现次数的方法，运行结果如图 3.22 所示。

代码段 3.47：执行嵌套的聚合，统计状态码在不同时间段中的数量

```
curl -H 'Content-Type: application/json' -XPOST localhost:9200/whale/log/
_search -d '{
  "aggs": {
```

```
        "logs_over_time": {
          "date_histogram": {
            "field": "timestamp",
            "interval": "3h",              //步长为 3h
            "format": "dd/MM/yyyy HH:mm:ss"          //根据 Elasticsearch 中的实际数据设置时间格式
          },
          "aggs": {
            "status_code": {
              "terms": {
                "field": "status_code",
                "order": {"_term": "asc"}
              }
            }
          }
        }
      }
    }'
```

```
▼ "aggregations": {
  ▼ "logs_over_time": {
    ▼ "buckets": [
        {
          "key_as_string": "25/05/2018 00:00:00",
          "key": 1527206400000,
          "doc_count": 61
        }
        ,
        {
          "key_as_string": "28/05/2018 00:00:00",
          "key": 1527465600000,
          "doc_count": 77
        }
        ,
        {
          "key_as_string": "31/05/2018 00:00:00",
          "key": 1527724800000,
          "doc_count": 73
        }
        ,
        {
          "key_as_string": "03/06/2018 00:00:00",
          "key": 1527984000000,
          "doc_count": 42
        }
      ]
    }
}
```

图 3.21　各个时间段的 date_histogram 聚合结果

```
{
    "key_as_string": "26/05/2018 00:00:00",
    "key": 1527292800000,
    "doc_count": 16,
  ▼ "status_code": {
      "doc_count_error_upper_bound": 0,
      "sum_other_doc_count": 0,
    ▼ "buckets": [
        {
          "key": "200",
          "doc_count": 11
        }
        ,
        {
          "key": "302",
          "doc_count": 2
        }
        ,
        {
          "key": "404",
          "doc_count": 1
        }
        ,
        {
          "key": "503",
          "doc_count": 2
        }
      ]
    }
}
```

图 3.22　每小时状态码出现次数的部分统计结果

6. filter 聚合

filter 聚合类似于 SQL 语句中的 where 子句的作用，可以为当前文档集合定义一个过

滤条件来缩小现有的数据集，凡满足定义的过滤条件(filter)的文档(document)都会被置入一个 bucket 中。代码段 3.48 展示了对类型文件 log 所有 size 字段大于 300 的文档进行平均值统计(在嵌套的聚合中实现)的方法，相当于满足指定条件后再进行的统计。针对类型文件 log 的运行结果如图 3.23 所示。

代码段 3.48：执行带有过滤的聚合

```
curl -H 'Content-Type: application/json' -XPOST localhost:9200/whale/log/
_search -d '{
  "aggs": {
    "filter_aggregation": {
      "filter": {
        "range": {
          "log_size": {"gt": 300}                    //规定日志信息长度大于 300
        }
      },
      "aggs": {
        "avg_log_size": {
          "avg": {"field": "log_size"}               //对聚合结果求平均值
        }
      }
    }
  }
}'
```

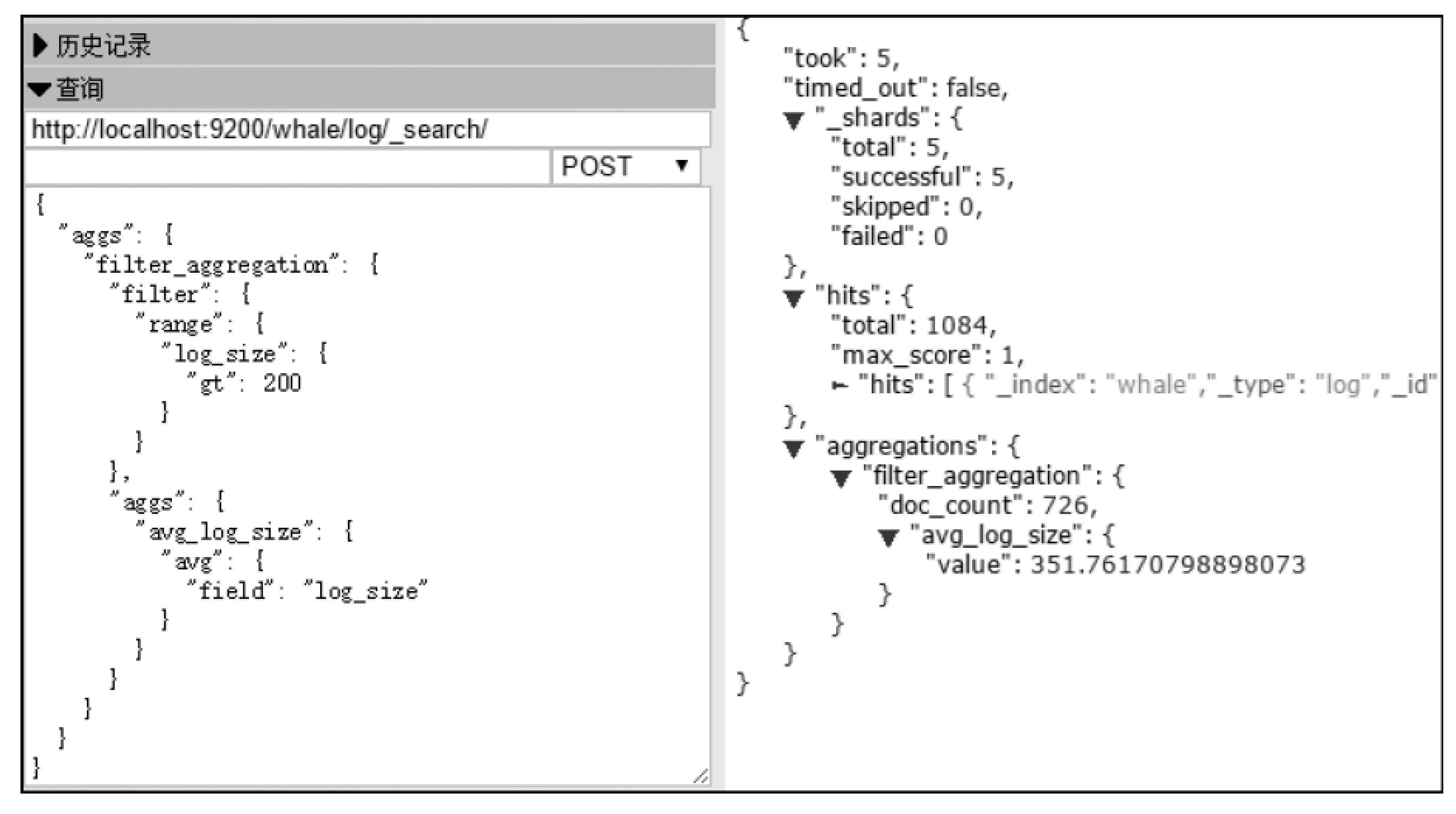


图 3.23 filter 聚合结果

3.4.3 pipeline 聚合

pipeline 聚合处理的对象是其他聚合的输出(而不是文档),这种聚合方式可向输出树添加信息。它包含很多形式,能够处理不同的任务,这些形式大致分为两类:

(1) parent:接收其父聚合的输出,并计算出新的 bucket 或新的聚合,添加到现有的 bucket 中。

(2) sibling:接收同级聚合的输出,并计算出新的同级聚合。

pipeline 聚合一般无子聚合,但是它可以在 buckets_path 中引用另一个 pipeline 聚合,这样 pipeline 聚合就可以被链接在一起。例如,可以将两个计算导数的 pipeline 聚合链接在一起以计算二阶导数。

本节将对 pipeline 聚合中的 min_bucket、max_bucket、sum_bucket、avg_bucket、stats_bucket、extended_stats_bucket 等聚合进行介绍。

1. min_bucket、max_bucket、sum_bucket、avg_bucket 聚合

min_bucket 和 max_bucket 聚合是上文提到的聚合的同级聚合,这样的聚合以同级聚合中特定指标的最小值来识别 bucket 并且输出 bucket 的值及其 key。这里的指标必须为数字类型,同级聚合必须是支持多个 bucket 的聚合。代码段 3.49 实现了对每小时内日志记录长度最小值的计算,结果如图 3.24 所示。

代码段 3.49: 计算每小时内日志记录长度最小值

```
curl -H 'Content-Type: application/json' -XPOST localhost:9200/whale/log/
_search -d '{
  "aggs": {
    "access_per_hour": {                    //外围聚合,统计每小时访问量
      "date_histogram": {
        "field": "timestamp",
        "interval": "hour"
      },
      "aggs": {                             //子聚合,统计 log_size 总数
        "access": {
          "sum": {"field": "log_size"}
        }
      }
    },
```

```
        "min_access_per_hour": {          //管道聚合,链接上面两种聚合
          "min_bucket": {
            "buckets_path": "access_per_hour>access"
          }
        }
      }
    }'
```

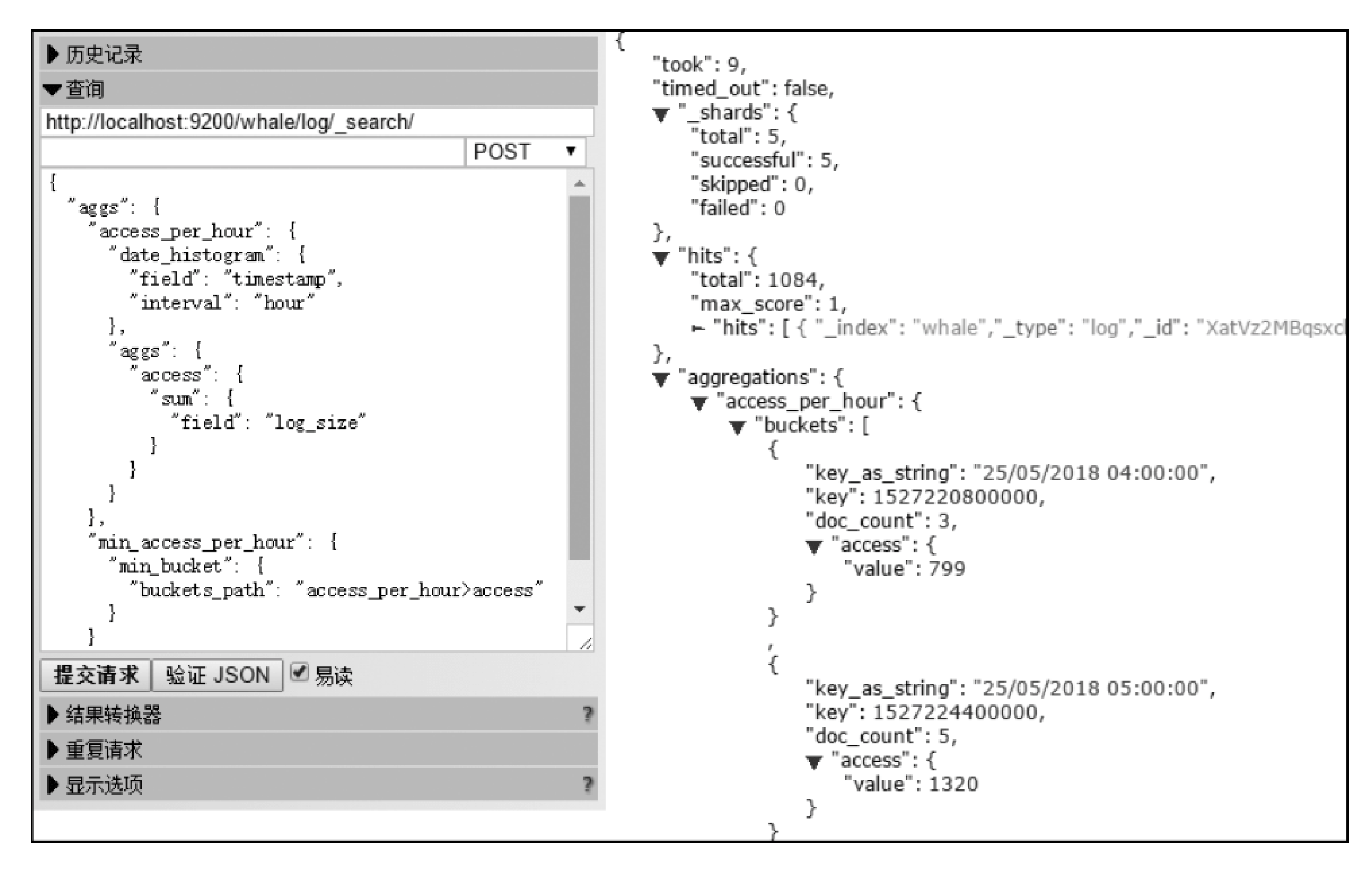


图 3.24　每小时内日志记录长度最小值计算结果

计算每小时内日志记录长度最大值时，只需将代码段 3.49 中的 min_bucket 改为 max_bucket 即可(可以与聚合的名字一并修改)，得到的结果如图 3.25 所示。如果使用 sum_bucket 和 avg_bucket 聚合，也采用这样的修改和执行方法，这里不再一一赘述。

2. stats_bucket、extended_stats_bucket 聚合

与上面提到的 4 种管道聚合类似，stats_bucket 聚合能够返回包含计数、最小值、最大值、平均值、和的多值计算结果。代码段 3.50 实现了每小时内日志记录长度的多值计算，结果如图 3.26 所示。



图 3.25 每小时内日志记录长度最大值计算结果

代码段 3.50：计算每小时内出现的日志记录长度计数、最小值、最大值、平均值、和

```
curl -H 'Content-Type: application/json' -XPOST localhost:9200/whale/log/
_search -d '{
  "aggs": {
    "access_per_hour": {                    //外围聚合，统计每小时访问量
      "date_histogram": {
        "field": "timestamp",
        "interval": "hour"
      },
      "aggs": {                             //子聚合，统计 log_size 总数
        "access": {
          "sum": {"field": "log_size"}
        }
      }
    },
    "status_access_per_hour": {             //管道聚合，链接上面两种聚合
      "stats_bucket": {
        "buckets_path": "access_per_hour>access"
      }
    }
  }
}'
```



图 3.26　每小时内日志记录长度的多值计算结果

extended_stats_bucket 聚合能够在 stats_bucket 聚合计算出的结果上添加平方和、方差和标准差等指标，只需将代码段 3.50 中的 stats_bucket 改为 extended_stats_bucket 即可（可以与聚合的名字一并修改），得到的结果如图 3.27 所示。



图 3.27　基于 extended_stats_bucket 聚合的每小时内日志记录长度的多值计算结果

3.4.4 matrix 聚合

matrix 聚合根据从查询的文档字段提取的值对多个字段进行操作，并返回矩阵结果。这一聚合方式与前面提到的 metric 聚合、bucket 聚合等均不同，它不支持 script 语句。本节对 matrix 聚合中的 matrix_stats 聚合进行介绍。

matrix_stats 聚合是一种面向数字的聚合，用于计算一组文档字段中的以下统计信息：

- 计数：计算过程中每种字段的样本数量。
- 平均值：每个字段数据的平均值。
- 方差：每个字段样本数据偏离平均值的程度。
- 偏度：每个字段样本数据在平均值附近的非对称分布情况的量化。
- 峰度：每个字段样本数据分布的形状的量化。
- 协方差：描述一个字段数据随另一个字段数据变化程度的矩阵。
- 相关性：描述两个字段数据之间的分布关系，其协方差矩阵取值为[－1,1]。

matrix 聚合主要用于计算两个数值型字段之间的关系。代码段 3.51 实现了对日志记录长度和 HTTP 状态码之间关系的计算，结果如图 3.28 所示。

代码段 3.51：计算日志记录长度和 HTTP 状态码之间的关系

```
curl -H 'Content-Type: application/json' -XPOST localhost:9200/whale/log/
_search -d '{
  "aggs": {
    "matrixstats": {
      "matrix_stats": {
        "fields": [
          "log_size",
          "status_code"
        ]
      }
    }
  }
}'
```

```
▼ "aggregations": {
  ▼ "matrixstats": {
      "doc_count": 1271,
    ▼ "fields": [
        {
            "name": "status_code",
            "count": 1271,
            "mean": 228.9095200629248,
            "variance": 6024.18944473011,
            "skewness": 2.643474511185891,
            "kurtosis": 8.689998822223084,
          ▼ "covariance": {
                "status_code": 6024.18944473011,
                "log_size": 388.6368889274364
            },
          ▼ "correlation": {
                "status_code": 1,
                "log_size": 0.03820842320486083
            }
        }
        ,
        {
            "name": "log_size",
            "count": 1271,
            "mean": 276.27694728560186,
            "variance": 17173.989381539737,
            "skewness": 0.025312169948124692,
            "kurtosis": 1.8053785042632087,
          ▼ "covariance": {
                "status_code": 388.6368889274364,
                "log_size": 17173.989381539737
            },
          ▼ "correlation": {
                "status_code": 0.03820842320486083,
                "log_size": 1
            }
        }
```


图 3.28 使用 matrix_stats 聚合计算 log_size 和 status_code 之间的关系

3.5 实例

前面给出了全文检索、词项检索等数据检索方法以及 metric、bucket 等聚合方法。通过这些方法，可以对索引库中的数据集进行检索并取得相关统计数据。下面的实例是对手机产品库数据集的检索与聚合，这些数据是利用专用的网络爬虫爬取到的，然后通过一定的方法建立索引并完成入库（利用 Java 采集数据存入 Elasticsearch 的过程详见 4.8 节）。手机产品库数据集的描述如下：

```
_index: yesky          //天极手机产品库数据的索引文件
_type: cellphone       //天极手机产品库数据的类型文件
_id: ×××               //id号
```

```
_version: ×                  //版本号
_score: ×                    //排序分值
_source: {                   //数据字段描述
    url: ×××                 //网址,如 http://product.yesky.com/product/877/877594/
    phoneName:×××            //手机名称和型号
    launchDate: ×××          //上市日期
    screenSize: ×××          //主屏幕尺寸 (in)
    resolution: ×××          //屏幕分辨率
    processor: ×××           //处理器名称
    battery: ×××             //电池容量 (mAh)
    ram: ×××                 //运行内存容量 (GB)
    rom: ×××                 //机身存储容量 (GB)
    backCamera: ×××          //主摄像头像素数 (万像素)
    frontCamera: ×××         //前摄像头像素数 (万像素)
}
```

首先在 Elasticsearch 中建立索引。在 head 的 Web 页面中,单击"复合查询"跳转至相应界面。在左侧"查询"窗口第一行填写 Elasticsearch 的 URL 地址,即 http://localhost:9200;在第二行填写要创建的索引名称,即 yesky,在右侧选择 PUT 方式;在第三行中输入创建索引的语句,如代码段 3.52 所示。单击"验证 JSON"按钮,经验证无误后即可提交请求。设置映像成功后,使用 Java 程序将在线数据采集至 Elasticsearch 中即可。

代码段 3.52: 为 yesky 索引设置映像

```
curl -H 'Content-Type: application/json' -XPUT localhost:9200/whale/_mapping/cellphone -d '{
                    //注意这里使用 PUT 方法
  "mappings": {
    "cellphone": {      //创建 cellphone 类型
      "properties": {
        "url": {
          "type": "keyword"
        },
        "phoneName": {
          "type": "text",
          "index": "analyzed",
          "analyzer": "ik_max_word"   //设置分析器
        },
```

```
            "price": {
              "type": "long"
            },
            "launchDate": {
              "type": "date",
              "format": "yyyy年 MM月 dd日"          //日期格式应根据实际数据设置
            },
            "screenSize": {
              "type": "keyword"
            },
            "resolution": {
              "type": "keyword"
            },
            "processor": {
              "type": "keyword"
            },
            "battery": {
              "type": "long"
            },
            "ram": {
              "type": "long"
            },
            "rom": {
              "type": "long"
            },
            "backCamera": {
              "type": "long"
            },
            "frontCamera": {
              "type": "long"
            }
          }
        }
      }
    }'
```

在爬取的数据中，可以利用手机的品牌或型号对手机产品数据进行全文检索。在图 3.29 中，使用全文检索中的 match_phrase_prefix 对手机产品信息进行查询。

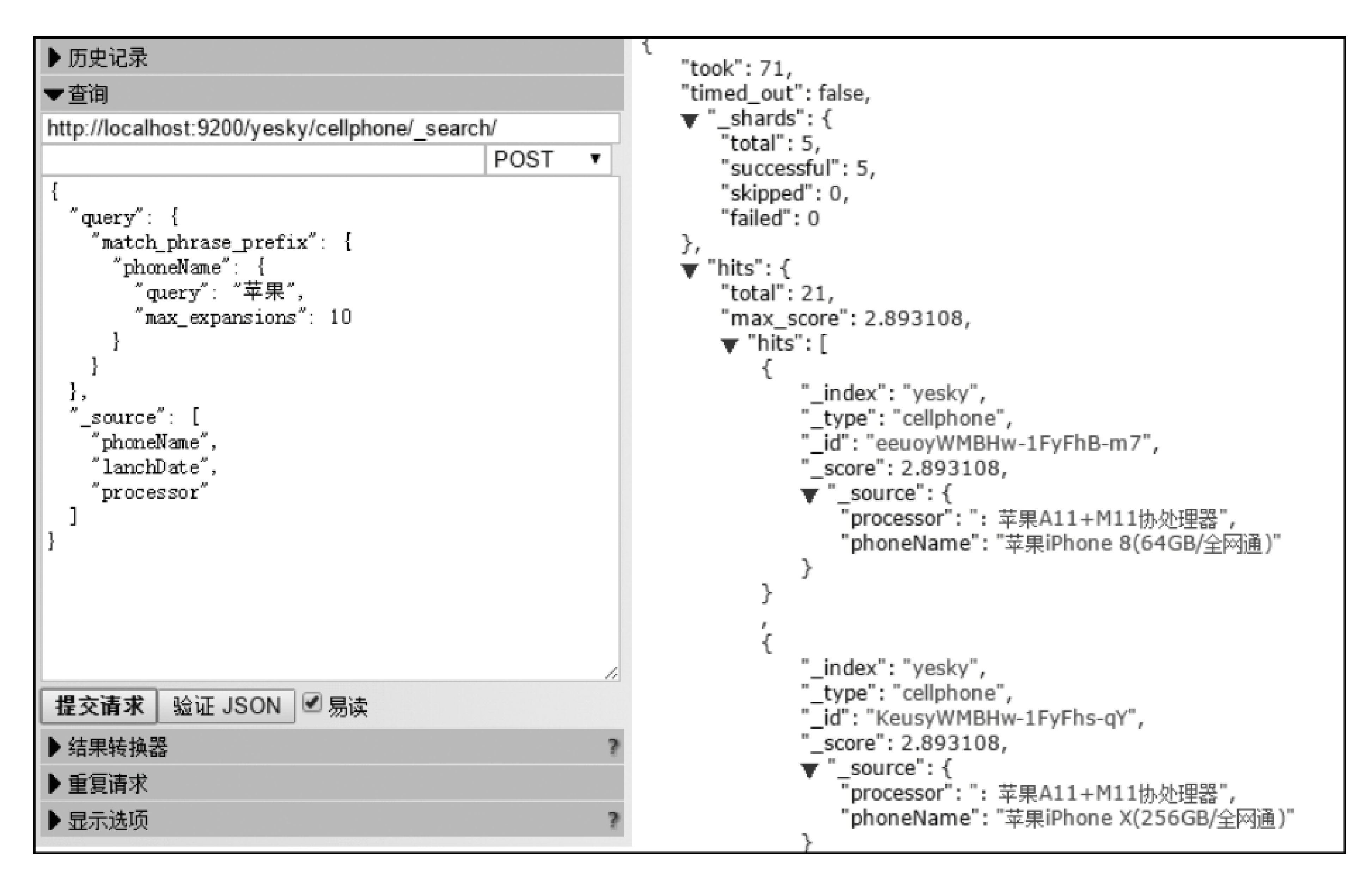


图 3.29　match_phrase_prefix 查询

采用词项检索中的 range 查询，对上市时间在特定日期范围内的手机进行检索，其检索的代码和结果如图 3.30 所示。



图 3.30　range 查询的实现

基于 Lucene 的 more_like_this 查询可以检索与指定内容相似的文档结果集。这种查询方法也可以用来实现初步的信息推荐。例如，利用 more_like_this 方法，在手机产品数据集中的 phoneName 字段检索与“vivo Z1(4GB/64GB/全网通)”相似的数据集，其代码和结果如图 3.31 所示。



图 3.31　more_like_this 查询的实现

聚合是十分有力的数据统计工具，能够使用户对数据集有宏观的了解。下面使用 metric 聚合中的 extended_stats 聚合对所有手机产品的售价进行统计，其代码和聚合结果图 3.32 所示。

可使用 bucket 聚合中的 histogram 聚合为柱状图的生成提供元数据。下面针对不同价位的手机进行 1000 元一档的价位统计，相关代码和结果如图 3.33 所示。

下面使用 matrix 聚合中的 matrix_stats 聚合对手机产品数据集中机身存储容量(rom 字段)和手机售价(price 字段)之间的关系进行矩阵计算。相关代码可参照代码段 3.49，结果如图 3.34 所示。

▶ 历史记录
▼ 查询
http://localhost:9200/yesky/cellphone/_search/
POST

```
{
  "aggs": {
    "price_status": {
      "extended_stats": {
        "field": "price"
      }
    }
  }
}
```

提交请求 验证 JSON ☑ 易读
▶ 结果转换器 ?
▶ 重复请求
▶ 显示选项 ?


```
{
  "took": 2,
  "timed_out": false,
  ▼ "_shards": {
    "total": 5,
    "successful": 5,
    "skipped": 0,
    "failed": 0
  },
  ▼ "hits": {
    "total": 151,
    "max_score": 1,
    ► "hits": [ { "_index": "yesky","_type": "cellphone","_id": "f
  },
  ▼ "aggregations": {
    ▼ "price_status": {
      "count": 151,
      "min": 0,
      "max": 8848,
      "avg": 3154.0132450331125,
      "sum": 476256,
      "sum_of_squares": 2051761516,
      "variance": 3640025.0594272185,
      "std_deviation": 1907.8849701769807,
      ▼ "std_deviation_bounds": {
        "upper": 6969.783185387074,
        "lower": -661.7566953208488
      }
    }
  }
}
```


图 3.32 extended_stats 聚合的实现

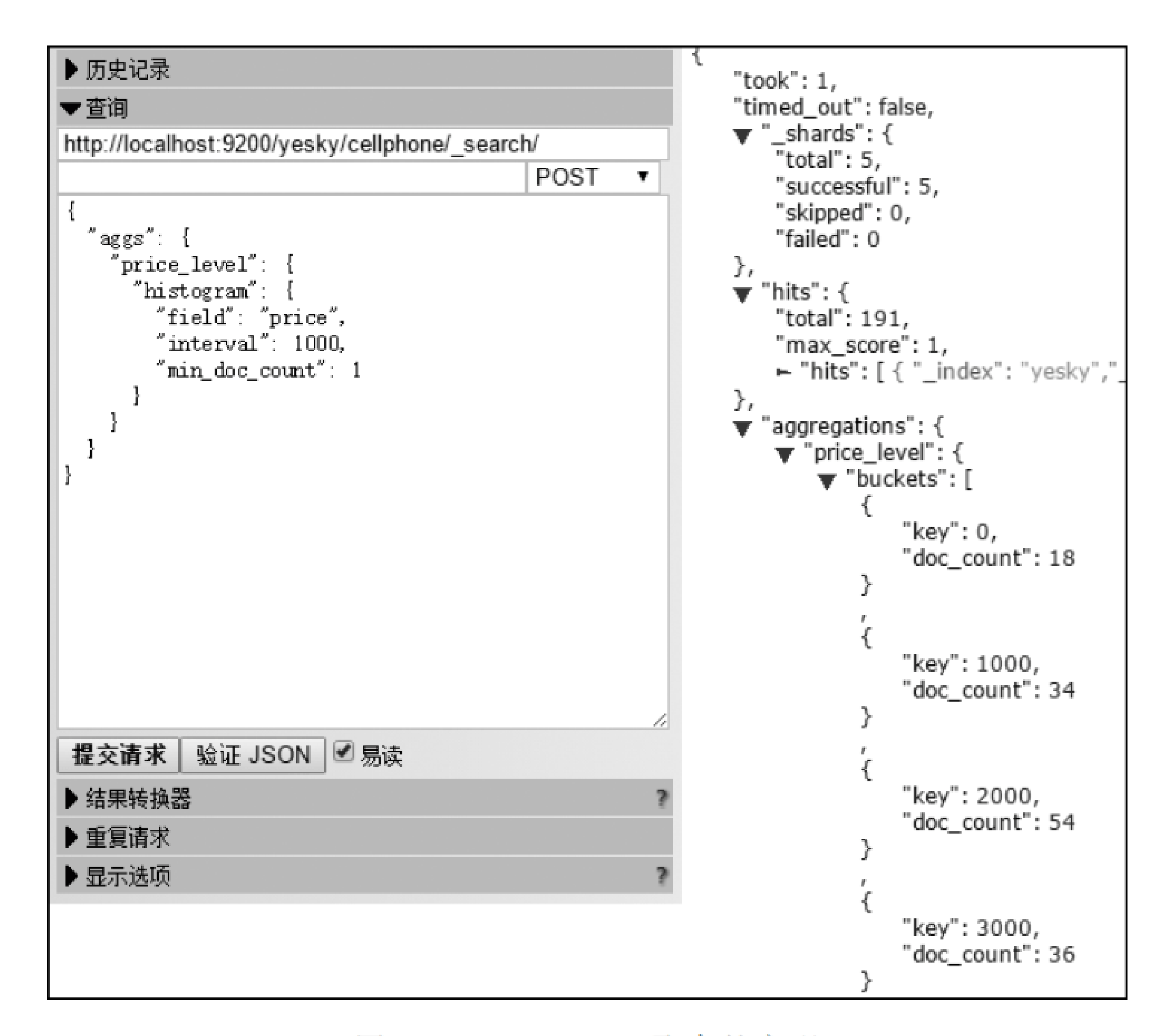


图 3.33 histogram 聚合的实现

```
▼ "aggregations": {
  ▼ "matrixstats": {
      "doc_count": 191,
    ▼ "fields": [
        {
          "name": "rom",
          "count": 191,
          "mean": 100.69633507853403,
          "variance": 4951.307302287131,
          "skewness": 1.9878585065895875,
          "kurtosis": 8.903779524498852,
        ▼ "covariance": {
            "rom": 4951.307302287131,
            "price": 85432.9826122899
          },
        ▼ "correlation": {
            "rom": 1,
            "price": 0.5784376726298753
          }
        }
        ,
        {
          "name": "price",
          "count": 191,
          "mean": 3337.994764397906,
          "variance": 4405732.405235602,
          "skewness": 1.213528066439307,
          "kurtosis": 5.4871879151394225,
        ▼ "covariance": {
            "rom": 85432.9826122899,
            "price": 4405732.405235602
          },
        ▼ "correlation": {
            "rom": 0.5784376726298753,
            "price": 1
          }
        }
      ]
```


图 3.34　matrix_stats 聚合的实现

3.6　扩展知识与阅读

有关 Lucene 的检索可参考文献(Michael,2011)。传统的 Web 服务是基于 RPC (Remote Procedure Call,远程过程调用)风格的,其实现技术主要包含 SOAP、WS(Web Service)标准栈等。RPC 风格的 Web 服务应用在分布式、开放的环境中会带来一些问题,如技术架构复杂、可伸缩性差等。而 RESTful 风格的 Web 服务可以解决上述问题。有关 RESTful 的内容,可以参阅文献(韩陆,2014)。文献(Rafal,2015)对 Elasticsearch 的搜索有更详细的说明,除此之外,还对扩展结构与搜索进行了说明。文献(余晟,2012)总结出一套使用正则表达式解题的办法,并通过具体的例子说明其实际应用,文中提到的各种统计是可以应用在实际的信息检索系统中的。文献(罗刚,2014)总结了搜索引擎相关理论与实际解决方案,包括搜索提示等内容,并给出了 Java 实现。在完成搜索提示时,很多时候可能会用到 Ajax 技术。文献(李刚,2014)介绍了 jQuery 1.8、Ext JS 4.1、Prototype 1.7.1、DWR 这几个常用的 Ajax 框架的用法,针对每个框架提供了实用方法。

3.7　本章小结

本章对基于 RESTful 的 Elasticsearch 信息检索方法进行了说明，分别从基本检索、结果过滤、复合查询等多个方面进行了说明，并给出了部分实际运行效果。另外，基于 metric、bucket、pipeline 和 matrix 机制的聚合统计分析功能日趋完善，可以使用各种脚本命令，也能将不同的聚合链接在一起工作。聚合的统计结果又可以由可视化工具加工处理成可视化的结果提供给用户，可进一步提升用户的搜索体验。



Elasticsearch API 及其应用

> All Elasticsearch operations are executed using a Client object. All operations are completely asynchronous in nature (either accepts a listener, or returns a future). Additionally, operations on a client may be accumulated and executed in Bulk.
>
> http://www.elastic.co

前面已经对 Elasticsearch 的索引、检索、统计等功能进行了说明。一般来说，对 Elasticsearch 中的信息进行检索等处理的大致步骤是：基于 analyzer 分析结果构建 query、给后台的 Elasticsearch 集群发送 query、完成检索并返回结果集合。其实，Elasticsearch 不仅可以通过前述的 RESTful 等方式进行操作，通过各种语言的 API（如 Java、Python、Ruby、PHP 等）也可以做所有的操作，其本质是各个语言的客户端封装底层的 HTTP 请求，调用 Elasticsearch 的 RESTful 接口。本章以应用较为广泛的 Java 为基础，重点介绍 Elasticsearch API Client 相关功能的 Java 实现，并在其中穿插少量 Python 客户端实现的内容。

4.1 Elasticsearch 节点实例化

在进行 Elasticsearch 的客户端编程时，首先要对 Elasticsearch 的节点进行实例化。在此之前，需要在 Java 工程中添加相关的 Elasticsearch 依赖（可以通过 Maven 添加，也可以手动导入已有的 JAR 包）；在 Python 环境中，需要通过 pip 包管理工具安装相关环境。

4.1.1 在 Java 中初始化 Elasticsearch

Maven 是基于项目对象模型、通过描述信息来管理项目的项目管理工具。Maven 一般包含项目对象模型、一组标准集合、项目生命周期、依赖管理系统以及运行逻辑等组成部分。

Maven 可以应用一些来自一组共享的(或者自定义的)逻辑插件。使用 Maven 时,可以用一个明确定义的项目对象模型来描述项目。Maven 的定义包括发布项目信息的方式以及一种在多个项目中共享 JAR 的方式(百度百科,2014a)。

> Tips 作为软件项目管理和理解工具,Maven 除了具备 Ant(一个将软件编译、测试、部署等步骤联系在一起的工具)的功能外,还使用项目对象模型对软件项目进行管理。它内置了更多的隐式规则,使得构建文件更加简单,内置依赖管理和 repository 来实现对依赖的管理和统一存储,并内置软件构建的生命周期。

POM 是 Maven 对一个单一项目的描述,它实现了一种以模型来描述的构建方式。图 4.1 显示的是在 IDEA 开发环境中基于 Maven 构建的工程中 pom. xml 的位置。

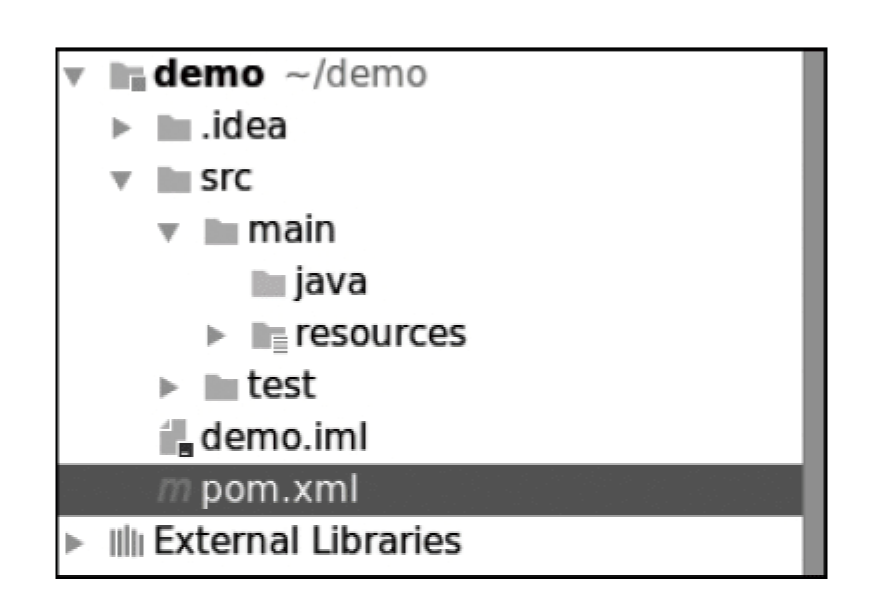


图 4.1 项目中的 pom. xml 文件

可以在 Java 工程中的 pom. xml 文件中添加和 Elasticsearch 相关的语句,以便将 Elasticsearch 的相关 JAR 包文件引入到相应的工程中。代码段 4.1 是一个 pom. xml 文件的内容,使用的 Elasticsearch 版本号为 6.2.4。

代码段 4.1: pom.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.
        apache.org/xsd/maven-4.0.0.xsd">
<modelVersion>4.0.0</modelVersion>

<groupId>com.cy</groupId>
<artifactId>demo</artifactId>
<version>1.0-SNAPSHOT</version>
```

```
<name> demo Maven Webapp</name>
<url> http://maven.aliyun.com</url>
<dependencies>
<dependency>
<groupId> org.elasticsearch.client</groupId>
<artifactId> transport</artifactId>
<version> 6.2.4</version>
</dependency>
<dependency>
<groupId> org.apache.logging.log4j</groupId>
<artifactId> log4j-api</artifactId>
<version> 2.6.2</version>
</dependency>
<dependency>
<groupId> org.apache.logging.log4j</groupId>
<artifactId> log4j-core</artifactId>
<version> 2.6.2</version>
</dependency>
</dependencies>
<build>
<finalName> demo</finalName>
</build>
</project>
```

将 pom. xml 写好之后，开发平台会自动下载这些依赖 JAR 包。

Tips 如果发现 Maven 在导入依赖过程中出现网络传输、程序错误等问题，可以尝试访问 pom. xml 指定的 URL 并从中直接下载 JAR 包，手动导入到开发平台的 library 中。

接下来需要初始化 Elasticsearch 的 TransportClient，这是一种轻量级的方法，它通过 Socket 与 Elasticsearch 集群相连，是基于 Netty 线程池的方式。在 TransportAddress 的构造方法内需填写一个已经启动的 Elasticsearch 节点的主机地址及端口号(默认 Transport 端口号是 9300)。可通过链式调用 addTransportAddress 方法添加很多类似节点，代码段 4.2 给出了实现方法。本章后续的大多数代码段中有基于 TransportAddress 的构造方法的实现。

代码段 4.2：基于 TransportClient 初始化客户端

```
//import 语句,略
//启动一个客户端
TransportClient client= new PreBuiltTransportClient(Settings.EMPTY)
    .addTransportAddress(new TransportAddress(InetAddress.getByName
    ("localhost"), 9300));
client.close();        //关闭客户端
```

在 Elasticsearch 安装主目录下的 config 文件夹下的 elasticsearch. yml 中提供了自定义集群名称的配置项 cluster. name。配置集群名称后，应在 Java 代码中指定集群名称，在创建 TransportClient 的过程中引入包含集群名称的 Settings，代码段 4. 3 给出了实现方法。其中，put()方法中的第一个参数是固定的，表示设置的是集群名称；第二个参数是指定的集群名称。

代码段 4.3：通过指定集群名称的方法来构建 TransportClient

```
//import 语句,略
Settings settings= Settings.builder()
    .put("cluster.name", "myClusterName").build();        //指定集群名称
TransportClient client= new PreBuiltTransportClient(settings);
//todo: 添加 Transport 地址,然后可以用客户端做其他事情
```

> Tips 一旦在 elasticsearch. yml 中配置了自定义的集群名称，在 Java 代码中就必须通过 Settings 类的 put() 方法指定集群名称，否则程序将无法找到符合配置的 Elasticsearch 节点。

还可通过 client. transport. sniff 方法开启嗅探模式来自动加入集群，如代码段 4. 4 所示。

代码段 4.4：开启嗅探模式

```
//import 语句,略
Settings settings= Settings.builder()
    .put("client.transport.sniff", true).build();
//启动一个客户端
TransportClient client= new PreBuiltTransportClient(settings);
client.close();        //关闭客户端
```

4.1.2 在 Python 中初始化 Elasticsearch

在 Python 环境中可利用 pip(一个通用的 Python 包管理工具，提供了对 Python 包的查找、下载、安装、卸载的功能)进行安装。首先需要确认是否安装过 pip 以及 pip 的版本，可在命令行窗口输入以下命令检查 pip 的版本：

```
pip show pip
```

如图 4.2 所示，命令行输出关于 pip 的信息，表示 pip 可正常运行且版本为 10.0.1。

```
C:\Users\84902>pip show pip
Name: pip
Version: 10.0.1
Summary: The PyPA recommended tool for installing Python packages.
Home-page: https://pip.pypa.io/
Author: The pip developers
Author-email: python-virtualenv@groups.google.com
License: MIT
Location: e:\programdata\anaconda3\lib\site-packages
Requires:
Required-by:
```

图 4.2 pip 及其版本信息

pip 的一个特点是可以通过命令行实现对第三方包的管理，可使用 pip -h 对所有命令进行查看。这里所安装的 Elasticsearch 是官方提供的 low-level 客户端，为了保证有更好的扩展性，使用 Python 语言实现了所有关于 Elasticsearch 的操作，它的操作方式更加接近于 Elasticsearch JSON DSL 的查询方式。使用 pip 安装 Elasticsearch 的命令如下：

```
pip install elasticsearch
```

上面的命令是安装 Python 6.2.x的客户端。如果系统安装的是 Elasticsearch 5.0，则需要指定安装客户端的版本，其命令如下：

```
pip install elasticsearch==5.0
```

安装完成后，可使用 pip show elasticsearch 命令查看已安装的 Elasticsearch 客户端信息。

> Tips 如果发现 pip 在安装过程中因为网络原因下载缓慢或下载失败，可切换国内镜像源加速访问。下面是常用的国内镜像源以及 pip 命令：
>
> ```
> pip install -i https://pypi.tuna.tsinghua.edu.cn/simple elasticsearch    //清华镜像
> pip install -i http://pypi.douban.com/simple/elasticsearch               //豆瓣镜像
> ```

接下来需要在 Python 中初始化 Elasticsearch 客户端，其实现方式是：将 Python 中的数据类型转换为 JSON，并基于 urlib3（功能强大的用于 HTTP 客户端请求的 Python 库）对集群进行请求。如代码段 4.5 所示，在创建客户端时，可以不指定任何参数，使用默认配置通过 localhost:9200 连接集群。

代码段 4.5：使用默认配置创建集群

```
from elasticsearch import Elasticsearch
es_client=Elasticsearch()                    //默认连接 localhost:9200
```

也可以通过传入 hosts 参数进行连接。传入的参数类型有两种：一种是包含 IP 地址和端口的字符串类型；另一种是列表类型，列表内的元素可以是 IP 地址和端口的字符串，也可以是包含 IP 地址和端口的字典。代码段 4.6 演示了这 3 种传入参数的方式。

代码段 4.6：指定 host 连接

```
from elasticsearch import Elasticsearch
es_client=Elasticsearch(hosts="localhost:9200")        //使用包含 IP 地址和端口的字符串形式
es_client=Elasticsearch(hosts=["localhost:9200"])
                                  //使用列表元素为包含 IP 地址和端口的字符串形式
es_client=Elasticsearch(hosts=[{"host": "localhost", "port": 9200}])
//使用列表元素为包含 IP 地址和端口的字典形式，注意：这里的端口号应为整数
```

同样，Python 也可以开启嗅探模式自动加入集群，需要在 Elasticsearch 客户端实例化时添加 3 个参数。代码段 4.7 开启嗅探模式。

代码段 4.7：开启嗅探模式

```
from elasticsearch import Elasticsearch
es_client=Elasticsearch(sniff_on_start=True,             //开启嗅探模式
                        sniff_on_connection_fail=True,   //嗅探失败重连
                        sniff_timeout=60)                //嗅探超时时间
```

4.2 索引数据

本节介绍创建索引的方法。先从准备 JSON 数据开始。

4.2.1 准备 JSON 数据

JSON 数据的产生有多种方法。

方法 1：产生符合 JSON 规范的字符串，并将其存于 String 或 byte[]类型的变量中。代码段 4.8 就是一个通过手工方法构建 JSON 字符串的示例。

代码段 4.8：通过手工方法构建 JSON 字符串

```
String json= "{"+ "\"name\":\"Tom\","+ "\"time\":\"2016-12-25\","+ "\"content\":\"happy\""+ "}";
```

方法 2：使用第三方的 JSON 工具库（如 Gson、Jackson 等）来序列化 JavaBean。

Tips Gson 是 Google 公司发布的一个开放源代码的 Java 库，主要用途是将 Java 对象序列化为 JSON 字符串，或将 JSON 字符串反序列化为 Java 对象。

在 Gson 实现过程中，首先在 pom.xml 中添加对 Gson 的依赖，见代码段 4.9。

代码段 4.9：在 pom.xml 中添加对 Gson 的依赖

```
<dependency>
    <groupId>com.google.code.gson</groupId>
    <artifactId>gson</artifactId>
    <version>2.8.5</version>
</dependency>
```

添加依赖完成之后，对需要处理的数据进行 JavaBean 序列化，见代码段 4.10 的 new Gson().toJson()语句：

代码段 4.10：序列化

```
//import 语句，略
Map<String, Object> map= new HashMap<> ();
map.put("name","hebust");
map.put("age",23);
map.put("content","hello world");
map.put("haa",new String[]{"big data","mining","information retrieval"});
String s= new Gson().toJson(map);          //对 map 中的数据进行 JavaBean 序列化
System.out.println(s);
```

方法 3：使用 Elasticsearch 自带的工具 XContentFactory.jsonBuilder()。代码段 4.11 是使用这个 JSON 工具类进行的操作，运行结果如图 4.3 的下部所示。所有格式的数据都会被转换为 byte[]格式。如果待处理的数据正是这样的格式，就可以直接使用它而不用再转换。

代码段 4.11：基于 XContentFactory.jsonBuilder()的方法

```
//import 语句,略
XContentBuilder builder= jsonBuilder().startObject()
      .field("name", "Tom")
      .field("time", new Date())
      .field("content", "happy")
      .endObject();
System.out.println(builder.string());                    //显示生成的 JSON字符串
```

```
        XContentBuilder builder= null;
        try {
            builder = jsonBuilder().startObject()
                    .field("name","Tom")
                    .field("time",new Date())
                    .field("content","happy")
                    .endObject();
            System.out.println(builder.string());
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
Run  ClientTest
/usr/jdk1.8.0_112/bin/java ...
{"name":"Tom","time":"2016-12-29T13:23:05.419Z","content":"happy"}

Process finished with exit code 0
```

图 4.3　JSON 数据准备

有关 jsonBuilder()的使用，在 4.3.3 节亦有体现。

Tips　Python 中有自带的 JSON 工具包，可以很方便地将 Python 中的字典、列表等数据类型进行序列化和反序列化。下面是将 Python 的数据序列化为 JSON 字符串的方法：

```
json_str= {"name":"Tom","time":"2016- 12- 25","content":"happy"}
print(json.dumps(json_str))
```

4.2.2　为 JSON 数据生成索引

当为 JSON 格式的文档生成索引时，可采用在 IndexResponse 中包含 Elasticsearch 索引信息的方法。



常用的准备索引 JSON 数据的 API 如下：

- prepareIndex()。
- prepareIndex(String index，String type)。
- prepareIndex(String index，String type，String id)。

代码段 4.12.1 给出生成 JSON 数据索引的部分代码，执行这段代码之前需要初始化 TransportClient，其中，要生成索引的内容是代码段 4.11 中提到的通过 XContentBuilder 生成的 JSON 数据，myweibo3 是在 Elasticsearch 中已经建立好的索引文件。

代码段 4.12.1：为 JSON 数据生成索引

```
//import 语句,略
TransportClient client= new PreBuiltTransportClient(Settings.EMPTY)
    .addTransportAddress(new TransportAddress
    (InetAddress.getByName("localhost"), 9300));
IndexResponse response= client.prepareIndex("myweibo3", "_doc")
    .setSource(jsonBuilder().startObject()
        .field("user", "cy")
        .field("post_date", "2018-01-01T18:00:00")
        .field("mymessage", "Java client test")
        .endObject()
    ).get();
//下面的语句是在控制台输出的索引信息
String _index= response.getIndex();          //得到索引文件名称
String _type= response.getType();            //得到类型文件名称
String _id= response.getId();                //得到文档 id 号
long _version= response.getVersion();        //版本号。若首次为该文档生成索引,则为 1
System.out.println(_index+ "\t"+ _type+ "\t"+ _id+ "\t"+ _version+ "\n");
client.close();                              //关闭客户端
```

针对代码段 4.12.1 的运行结果如图 4.4 所示，其中由代码段 4.12.1 输出的结果在图 4.4 的下部。相应地，Elasticsearch 产生的索引数据如图 4.5 所示。

代码段 4.12.2 提供了为 JSON 数据生成索引的 Python 实现方式，该方式简单易懂，且大多数 Python 客户端的实现方式也只是在 body 查询语句上有所区别，查询语句也类似于第 3 章中的内容，因而后续 Python 客户端所实现的内容以本段代码为基础模板，当与 Java 的实现方式有较大差异时，则会进行说明。其他实现方式可参考第 3 章的相关内容。

```
IndexResponse response = client.prepareIndex( index: "myweibo3",  type: "_doc")
        .setSource(jsonBuilder().startObject()
                .field( name: "user",  value: "cy")
                .field( name: "post_date",  value: "2018-01-01T18:00:00")
                .field( name: "mymessage",  value: "Java client test")
                .endObject()
        ).get();
//下面的语句是在控制台输出得到的索引信息
String _index = response.getIndex();            //得到index 名称
String _type = response.getType();              //得到type 名称
String _id = response.getId();                  //得到document id
long _version = response.getVersion();          // 版本号。如是首次索引该文档，则此值为1
System.out.println(_index + "\t" + _type + "\t" + _id + "\t" + _version + "\n");
client.close();
```

```
Run  ClientTest
/usr/jdk1.8.0_112/bin/java ...
myweibo3    _doc    DS_VxGMBmDMkTsldbzzl    1

Process finished with exit code 0
```

图 4.4　代码段 4.12.1 输出的结果

```
{
    "_index": "myweibo3",
    "_type": "_doc",
    "_id": "DS_VxGMBmDMkTsldbzzl",
    "_version": 1,
    "_score": 1,
    "_source": {
        "user": "cy",
        "post_date": "2018-01-01T18:00:00",
        "mymessage": "Java client test"
    }
}
```

图 4.5　Elasticsearch 产生的索引数据

代码段 4.12.2：在 Python 下为 JSON 数据生成索引

```
from elasticsearch import Elasticsearch
es_client=Elasticsearch()      //创建客户端,若不指定参数,则默认为 localhost:9200
body={                         //创建查询主体,设置查询语句
    "query": {
        "term": {
            "user": "cy"
        }
    }
}
res=es_client.search(index="myweibo3", doc_type="_doc", body=body)    //通过 search 语句进行检索
for hit in res['hits']['hits']:        //输出索引数据
    print(hit['_source'])
```

4.3 对索引文件的操作

4.3.1 获取索引中的文档数据

前面已经介绍了如何在 Elasticsearch 中获取待处理的文档数据。其实，在 Get API 的帮助下，也可以在 Java 客户端获取文档数据。代码段 4.13 给出了方法，请注意其中的 prepareGet(String index, String type, String id) 方法，其中的 3 个参数分别是 Elasticsearch 中的某个索引文件名、某个类型文件名、文档的 id 号（为便于说明，代码中直接给出了某个文档的 id 号）。

代码段 4.13：获取文档数据

```
//import 语句，略
TransportClient client= new PreBuiltTransportClient(Settings.EMPTY)
     .addTransportAddress(new TransportAddress(InetAddress.getByName
     ("localhost"), 9300));
GetResponse response= client.prepareGet("information", "_doc", "1").get();
Map< String, Object> hit= response.getSource();
String _title= hit.get("Title").toString();        //显示针对该条的数据细节
String _author= hit.get("author").toString();
String _abstract= hit.get("abstract").toString();
String _keywords= hit.get("keywords").toString();
System.out.println(_title+ "\n"+ _author+ "\n"+ _abstract+ "\n"+ _keywords);
client.close();
```

图 4.6 和图 4.7 给出了实际运行结果，其中，图 4.6 是索引文件 information 中某个文档的内容，图 4.7 是程序运行结果。

```
{
    "_index": "information",
    "_type": "_doc",
    "_id": "1",
    "_version": 1,
    "_score": 1,
    "_source": {
      "title": "深度学习",
      "author": "Ian Goodfellow，Yoshua Bengio，Aaron Courville",
      "abstract": "《深度学习》由全球知名的三位专家Ian Goodfellow、Yoshua Bengio和Aaron Courville撰写，是深度学习领域奠基性的经典教材。全书的内容包括3个部分：第1部分介绍基本的数学工具和机器学习的概念，它们是深度学习的预备知识；第2部分系统深入地讲解现今已成熟的深度学习方法和技术；第3部分讨论某些具有前瞻性的方向和想法，它们被公认为是深度学习未来的研究重点。",
      "keywords": "深度学习 自然语言处理 计算机视觉 推荐系统"
    }
}
```

图 4.6 索引文件 information 中某个文档的内容

```
GetResponse response = client.prepareGet( index: "information",  type: "_doc",  id: "1").get();
Map<String, Object> hit = response.getSource();
String _title = hit.get("title").toString();                    //显示针对该条的数据细节
String _author = hit.get("author").toString();
String _abstract = hit.get("abstract").toString();
String _keywords = hit.get("keywords").toString();
System.out.println(_title + "\n" + _author + "\n" + _abstract + "\n" + _keywords);
client.close();
```

```
Run  ClientTest
/usr/jdk1.8.0_112/bin/java ...
深度学习
Ian Goodfellow, Yoshua Bengio, Aaron Courville
《深度学习》由全球知名的三位专家Ian Goodfellow、Yoshua Bengio和Aaron Courville撰写，是深度学习领域奠基性的经典
深度学习 自然语言处理 计算机视觉 推荐系统

Process finished with exit code 0
```

图 4.7　获取的文档数据

Elasticsearch 可以同时获取多个文档的数据，使用 MultiGet API 可以实现在不同索引文件中获取多个 id 号的内容。代码段 4.14 实现了获取两个索引文件中的多个文档数据的功能，结果如图 4.8 所示。

代码段 4.14：获取多个索引文件中的文档数据

```
//import 语句，略
TransportClient client= new PreBuiltTransportClient(Settings.EMPTY)
    .addTransportAddress(new TransportAddress(InetAddress.getByName
    ("localhost"), 9300));
MultiGetResponse multiGetItemResponses= client.prepareMultiGet()
    .add("myweibo3", "_doc", "3")
    //指定多个 id号
    .add("myweibo3", "_doc", "1m8gdGMB27N6d9FAnKf8", "DS_VxGMBmDMkTs1dbzz1")
    .add("information", "_doc", "3")
    .get();
for(MultiGetItemResponse itemResponse : multiGetItemResponses) {
    GetResponse response= itemResponse.getResponse();
    if(response.isExists()) {
      System.out.println(response.getSourceAsString());          //输出结果
    }
}
client.close();
```

```
MultiGetResponse multiGetItemResponses = client.prepareMultiGet()
        .add( index: "myweibo3",  type: "_doc",  id: "3")
        //指定多个id号
        .add( index: "myweibo3",  type: "_doc",  ...ids: "1m8gdGMB27N6d9FAnKf8",
            "DS_VxGMBmDMkTsldbzzl")
        .add( index: "information",  type: "_doc",  id: "3")
        .get();
for (MultiGetItemResponse itemResponse : multiGetItemResponses) {
    GetResponse response = itemResponse.getResponse();
    if (response.isExists()) {
        System.out.println(response.getSourceAsString());        //输出结果
    }
}
```

```
Run   ClientTest
/usr/jdk1.8.0_112/bin/java ...
{
"user":"LiMing",
"post_date":"2018-01-01T14:00:00",
"mymessage":"Hello Tom"
}
{
"user":"Alan",
"post_date":"2018-01-01T08:00:00",
"mymessage":"This is an example on creating an index"
}
{"user":"cy","post_date":"2018-01-01T18:00:00","mymessage":"Java client test"}
{"title":"大数据搜索与挖掘及可视化管理方案（第3版）","author":"高凯","abstract":"本书着重介

Process finished with exit code 0
```

图 4.8　获取多个文档的数据

Tips 常用的获取索引文件中的文档数据的 API 如下：

- get(GetRequest request)、get(GetRequest request, ActionListener<GetResponse> listener)：根据 GetRequest 提供的 index、type 和 id 获取文档数据。
- prepareGet()、prepareGet(String index, String type, String id)：准备获取，但不执行获取操作。
- multiGet(MultiGetRequest request)、multiGet(MultiGetRequest request, ActionListener<MultiGetResponse>listener)：批量获取文档数据。
- prepareMultiGet()：准备批量获取，但不执行获取操作。

Tips 如果使用 Python 从多个索引文件中获取文档数据，可在 es_client 检索时添加 index 参数，指定需要获取哪些索引文件中的数据：

```
es_client.search(index=[myweibo3, informatio],body=query)
```

4.3.2　删除索引文件中的文档数据

和获取文档信息 API 类似，也可以在 Java 客户端删除索引文件中的文档数据，可以通

过 prepareDelete()等方法实现，如代码段 4.15 所示。prepareDelete()方法的第三个参数是拟删除文档的 id 号，其前面两个参数则分别是索引文件名称、统一的类型文件名称。

代码段 4.15：删除文档信息

```
//import 语句,略
TransportClient client= new PreBuiltTransportClient(Settings.EMPTY)
      .addTransportAddress(new TransportAddress(InetAddress.getByName
      ("localhost"), 9300));
DeleteResponse response= client.prepareDelete("information", "_doc", "3").get();
int _status= response.status().getStatus();      //获取删除操作的状态码
String _index= response.getIndex();
String _type= response.getType();
String _id= response.getId();
System.out.println(_status+ "\t"+ _index+ "\t"+ _type+ "\t"+ _id);
client.close();
System.out.println("指定 id号的记录已经被删除");
```

 常用的删除索引文件中的文档数据的 API 如下：

- prepareDelete()、prepareDelete(String index, String type, String id)：准备删除，但不执行删除操作。
- delete(DeleteRequest request)、delete(DeleteRequest request, ActionListener <DeleteResponse> listener)：根据 DeleteRequest 提供的 index、type 和 id 从索引文件中删除文档数据。

Tips 在 Python 中可使用 delete 函数进行索引文件中的文档数据的删除操作，相关实现方式如下：

```
es_client.delete(index= "information", doc_type= "_doc", id= 3)
```

4.3.3 更新索引文件中的文档数据

和获取文档数据的方法类似，也可以在 Java 客户端更新索引文件中的文档数据。代码段 4.16 给出了实现方法，通过 Client 对象的 update()方法实现。注意，这里对指定 id 号的多个字段信息进行了更新操作。

代码段 4.16：更新文档数据，注意 jsonBuilder()的用法

```
//import 语句,略
TransportClient client= new PreBuiltTransportClient(Settings.EMPTY)
      .addTransportAddress(new TransportAddress(InetAddress.getByName
      ("localhost"), 9300));
UpdateRequest updateRequest= new UpdateRequest();
updateRequest.index("myweibo3");                    //指定索引文件
updateRequest.type("_doc");                         //指定类型文件
updateRequest.id("DS_VxGMBmDMkTsldbzzl");           //指定 id号
//到此为止,已定位到待更新的记录上。下面是对该记录中的部分字段值进行更新
updateRequest.doc(jsonBuilder()                     //使用 jsonBuilder()方法
      .startObject()
      .field("post_date", "2017-01-01T09:00:00")
      .field("mymessage", "Java client update test")
      .endObject());
client.update(updateRequest).get();                      //通过 client对象的 update()方法实现
client.close();
```

常用的更新索引文件中的文档数据的 API 如下：

- update(UpdateRequest request)：基于脚本更新文档并返回更新后的结果。
- update(UpdateRequest request, ActionListener<UpdateResponse> listener)：借助于监听器完成数据的更新。
- prepareUpdate()、prepareUpdate(String index, String type, String id)：准备更新，但不执行更新操作。

在 Python 中可使用 update()函数进行数据的更新操作，实现方式如下，其中 body 为要更新的内容。

```
es_client.update(index="myweibo3", doc_type="_doc", id=3,body=body)
```

4.3.4 对索引文件中的文档进行批量操作

可通过 bulk API 对指定的文档进行批量操作，示例方法如代码段 4.17.1 所示，这里通过 bulk 操作同时进行了两个向索引文件中增加数据的操作(prepareIndex)和一个更新操作(prepareUpdate)。

代码段 4.17.1：对文档的批量操作

```
//import 语句，略
TransportClient client= new PreBuiltTransportClient(Settings.EMPTY)
        .addTransportAddress(new TransportAddress(InetAddress.getByName
        ("localhost"), 9300));
BulkRequestBuilder bulkRequest= client.prepareBulk();
bulkRequest.add(client.prepareIndex("myweibo3", "_doc", "1")
        .setSource(jsonBuilder()
            .startObject()
            .field("user", "cy")
            .field("post_date", "2018-01-01T09:00:00")
            .field("mymessage", "bulk index 1")
            .endObject()
        )
);
bulkRequest.add(client.prepareIndex("myweibo3", "_doc", "2")
        .setSource(jsonBuilder()
            .startObject()
            .field("user", "cy")
            .field("post_date", "2018-01-01T09:00:00")
            .field("mymessage", "bulk index 2")
            .endObject()
        )
);
bulkRequest.add(client.prepareUpdate("myweibo3", "_doc", "DS_VxGMBmDMkT5ldbzzl")
        .setDoc(jsonBuilder()
            .startObject()
            .field("mymessage", "Java client bulk test")
            .endObject()
        )
);
BulkResponse bulkResponse= bulkRequest.get();
if(bulkResponse.hasFailures()) {
//可在这里对失败请求进行处理，略
}
client.close();
```



常用的对索引文件中的文档进行批量操作的 API 如下：

- bulk（BulkRequest request）、bulk（BulkRequest request， ActionListener <BulkResponse> listener）：批量操作。
- prepareBulk()：准备进行批量操作，但不执行。

在 Python 中对文档进行批量操作，需要首先导入 helpers，然后用多个 actions 将数据保存至列表中，最后使用 helpers. bulk 进行批量操作。代码段 4. 17. 2 实现了插入和更新操作。

代码段 4.17.2：对文档的批量操作

```
from elasticsearch import Elasticsearch
import elasticsearch.helpers
actions= []
es= Elasticsearch("localhost:9200")
actions.append({'_op_type': 'index', '_index': "myweibo3",'_type': "_doc", '_id': '1',
     '_source': {
          'user': 'cy',
          "post_date": "2018-01-01T09:00:00",
          "mymessage": "bulk index 1"
     }
})
actions.append({'_op_type': 'index','_index': "myweibo3", '_type': "_doc", '_id': '2',
     '_source': {
          'user': 'cy',
          "post_date": "2018-01-01T09:00:00",
          "mymessage": "bulk index 2"
     }
})actions.append({ '_op_type': 'update','_index': "myweibo3", '_type': "_doc",'_id': 'DS_
VxGMBmDMkTsldbzzl','doc': {
          "mymessage": "Java client bulk test"
     }
})elasticsearch.helpers.bulk(es, actions)        //进行批量操作
```

4.4 信息检索

信息检索是 Elasticsearch 的主要功能。Elasticsearch 提供了多种方式供前端实现信息检索功能。

4.4.1 概述

Elasticsearch 提供了强大的全文检索功能。借助相应的 API，可以使用 SearchSourceBuilder 构造一个 Elasticsearch 检索请求。一般地，首先要在项目中引入相应的类文件，例如：

```
import org.elasticsearch.action.search.SearchResponse;
import org.elasticsearch.action.search.SearchType;
import org.elasticsearch.index.query.FilterBuilders.* ;
import org.elasticsearch.index.query.QueryBuilders.* ;
```

之后，在 Elasticsearch Client 实例化对象的 prepareSearch()方法中设置索引文件的名称，在 setTypes()方法中设置统一的类型文件，在 setQuery()方法中设置查询，而这个查询可使用 Elasticsearch 自带的 QueryBuilders()方法来构建。当然，也可在 setPostFilter()方法中设置在搜索前需要执行的过滤操作(可使用 FilterBuilders()方法构建过滤器)，如代码段 4.18 所示。其中，client. prepareSearch 用来创建一个 SearchRequestBuilder，client. prepareSearch()方法的参数是一个或多个索引文件名称，这里的 setSearchType()中的参数是一个枚举类型的元素，包括两个可选择的值：

(1) QUERY_THEN_FETCH：先向所有的分片发出请求，各分片只返回排序和排名相关的信息(不包括文档)，然后按照各分片返回的分数进行排序并取前 size 个文档，之后再从相关的分片中取出文档。这对于有许多分片的索引来说是很便利的，因为返回结果不会重复。

(2) DFS_QUERY_THEN_FETCH：与 QUERY_THEN_FETCH 相似，不同之处在于：参数取值为 DFS_QUERY_THEN_FETCH 时，首先计算分布词频，以获得更准确的得分。

代码段 4.18：检索操作示例

```
//import 语句,略
TransportClient client= new PreBuiltTransportClient(Settings.EMPTY)
.addTransportAddress(new TransportAddress(InetAddress.getByName("localhost"), 9300));
SearchResponse response= client.prepareSearch("it-home")  //指定索引文件
        .setTypes("_doc")                                 //指定统一的类型文件
        .setSearchType(SearchType.DFS_QUERY_THEN_FETCH)
        .setQuery(QueryBuilders.termQuery("content", "java"))      //检索 content 字段中的词项 java
```

```
        .setPostFilter(QueryBuilders.rangeQuery("publishTime").from("2018-01-01").to("now"))
        .setFrom(0).setSize(60).setExplain(true)
        .get();
SearchHit[] hits=response.getHits().getHits();
for(SearchHit hit : hits) {
    System.out.println(hit.sourceAsString());
}
client.close();
```



常用的检索 API 如下：

- search(SearchRequest request)、search(SearchRequest request, ActionListener<SearchResponse> listener)：根据 SearchRequest 提供的 index、type 和 id 执行搜索操作。
- prepareSearch(String... indices)：准备搜索，但不执行搜索操作。

4.4.2 multiSearch

multiSearch 是 Elasticsearch 提供的针对多个查询请求进行一次查询的接口。该接口虽能同时执行多个不同的查询，但无法对最终结果自动分页，而且有可能多个查询请求得到的结果中存在重复，但 multiSearch 并不负责去重。代码段 4.19 展示了如何使用 Java 客户端进行 multiSearch 操作，首先在 prepareSearch()中构造两个查询（设定查询项及返回结果集大小），之后分别将它们添加到 prepareMultiSearch()中并执行多重查询。

```
代码段 4.19: multiSearch
//import 语句,略
TransportClient client=new PreBuiltTransportClient(Settings.EMPTY)
        .addTransportAddress(new TransportAddress(InetAddress.getByName
        ("localhost"), 9300));
SearchRequestBuilder srb1=client
        .prepareSearch().setQuery(QueryBuilders.queryStringQuery("json OR
        gson")).setSize(1);
SearchRequestBuilder srb2=client
        .prepareSearch().setQuery(QueryBuilders.matchQuery("content", "java")).
        setSize(1);
```

```
MultiSearchResponse sr= client.prepareMultiSearch()
        .add(srb1)
        .add(srb2)
        .get();
long nbHits= 0;
for(MultiSearchResponse.Item item : sr.getResponses()) {
        SearchResponse response= item.getResponse();
nbHits+ = response.getHits().getTotalHits();
}
System.out.println("共检索到"+ nbHits+ "条记录。");
client.close();
```



常用的执行多个搜索请求的 multiSearch API 如下：

- multiSearch(MultiSearchRequest request)。
- multiSearch(MultiSearchRequest request, ActionListener<MultiSearchResponse> listener)。
- prepareMultiSearch()。

4.4.3　查询模板

Elasticsearch 提供了查询模板接口，通过这一接口，可以将 JSON 格式的 RESTful 查询语句嵌入到 Java 代码中。在执行查询时，将键-值对形式的参数填充到语句中相应的位置执行。代码段 4.20.1 利用查询模板实现了 match 查询，由 setScript()方法添加 RESTful 查询语句，语句中{{param}}是一个形式参数，实际参数是通过将搜索词 java 放入键-值对中传入的。

代码段 4.20.1：在 Java 中利用查询模板执行 match 查询

```
//import 语句,略
TransportClient client= new PreBuiltTransportClient(Settings.EMPTY)
        .addTransportAddress(new TransportAddress(InetAddress.getByName
        ("localhost"), 9300));
Map<String, Object> template_params= new HashMap<String, Object>();         //传递参数的键值对
template_params.put("param", "java");                                        //实际参数放入键值对
SearchResponse response= new SearchTemplateRequestBuilder(client)
```

```
        .setScript("{\n"+                          //添加 RESTful 查询语句
            "        \"query\" : {\n"+
            "            \"match\" : {\n"+
            "                \"content\" : \"{{param}}\"\n"+        //给出形式参数
            "            }\n"+
            "        }\n"+
            "}")
        .setScriptType(ScriptType.INLINE)          //将查询语句和代码写在一起
        .setScriptParams(template_params)          //传入键-值对,用于传入实际参数
        .setRequest(new SearchRequest())
        .get()
        .getResponse();
SearchHit[] hits= response.getHits().getHits();
for (SearchHit hit : hits) {
System.out.println(hit.getSourceAsString());
}
client.close();
```

Python 中的模板查询语句与 3.2.6 节所述相似，但 Python 中使用检索 API 需要改为 search_template，代码段 4.20.2 实现了在 Python 中使用模板查询的方法。

代码段 4.20.2：在 Python 中利用查询模板执行 match 查询

```
//import 语句,略
es= Elasticsearch()
body= {
    "inline": {
        "query": {
            "match": {
                "{{my_field}}": "{{my_value}}" //设置形式参数,使用{{ }}括起来
            }
        },
        "size": "{{my_size}}"
    },
    "params": {
        "my_field": "content",              //设置实际参数
        "my_value": "java",
        "my_size": 5
    }
}
res= es.search_template(index= "it- home", body= body)        //注意,这里需要改为 search_template
for hit in res['hits']['hits']:
    print(hit['_source'])
```

4.4.4 Query DSL 概述

前面提到，Search API 允许执行一次信息检索，返回一个与查询匹配的结果集。它可以在一个（或多个）索引文件上执行。而本节介绍的 Query DSL 能够专注于特定问题，执行更为专业的检索任务。

Query DSL 分为全文检索、词项检索、复合查询、连接查询、跨度查询、特殊查询和脚本等内容。其中，全文检索包括 match 查询、multi_match 查询、simple_query_string 查询等，用于执行不同形式的全文检索任务；词项检索包括 term 查询、terms 查询、range 查询、prefix 查询、wildcard 查询、regexp 查询等，用于执行对不可拆分的关键词词项的检索；复合查询包括 bool 查询、boosting 查询等，能够根据用户对不同条件下的数据的需要，对数据进行综合信息检索的任务；跨度查询包括 span_term 查询、span_or 查询、span_containing 查询、span_within 查询等，能够根据不同形式的跨度（如时间、两词项截出的文本段）执行对应的查询；特殊查询包括 more like this 查询，它和脚本不属于上述任何一类，可以完成较为特殊的任务。

在 Java 客户端使用的 Query DSL 和 RESTful 是相似的。编程者可以用 QueryBuilder 来构建 query，然后使用 Query DSL 进行搜索。下面对基于 query 的客户端实现方法进行介绍。

4.4.5 matchAllQuery

Elastic 公司将 matchAllQuery()作为全文检索之外的查询方法，其作用是匹配文档中的所有字段。当需要匹配所有项或者查询某些索引文件下的记录总量时，可以使用 matchAllQuery()方法。它相当于关系数据库 SQL 语句中的 select * from table 语句，其后的 setFrom()和 setSize()用于控制返回的结果集大小。该方法的实现如代码段 4.21 所示。

代码段 4.21：使用 matchAllQuery()方法查询所有字段

```
//import 语句，略
TransportClient client= new PreBuiltTransportClient(Settings.EMPTY)
    .addTransportAddress(new TransportAddress(InetAddress.
    getByName("localhost"), 9300));
    QueryBuilder qb=matchAllQuery();
    SearchResponse response=client.prepareSearch("it-home")    //指定索引文件
    .setTypes("_doc")           //指定统一的类型文件名
    .setQuery(qb)               //指定 match all query
```

```
        .setFrom(0)
        .setSize(10)
        .get();
for(SearchHit hit : response.getHits().getHits()) {          //遍历检索结果
        System.out.println(hit.getSourceAsString());
}
client.close();
```

4.4.6 全文检索

1. matchQuery()

全文检索方法 matchQuery(String name, Object text)能够使用某一字段的值对文档进行检索。代码段 4.22 实现了 matchQuery()的功能,在索引文件名为 it-home 的文档中,在 content 字段搜索包含“程序员”字符串的数据集。其他的相关方法,如 setSearchType()、setFrom()、setSize()、setExplain()等,不再给出解释与使用说明。可以将 matchQuery()定义为一个 QueryBuilder()方法的实例,传入 setQuery()方法中。

```
代码段 4.22: 使用 matchQuery()方法实现检索
//import 语句,略
TransportClient client= new PreBuiltTransportClient(Settings.EMPTY)
addTransportAddress(new TransportAddress(InetAddress.getByName("localhost"), 9300));
QueryBuilder qb= matchQuery(
        "content",                                      //字段
        "程序员"                                        //搜索词
);
SearchResponse response= client.prepareSearch("it-home")
        .setTypes("_doc")
        .setSearchType(SearchType.DFS_QUERY_THEN_FETCH)
        .setQuery(qb)
        .setFrom(0)
        .setSize(10)
        .setExplain(true)
        .get();
for(SearchHit hit : response.getHits().getHits()) {          //遍历检索结果
        System.out.println(hit.getSourceAsString());
}
client.close();
```

> Tips matchQuery()能够使用某一字段的值对文档进行检索；matchAllQuery()用于匹配文档中的所有字段。

2. multiMatchQuery()

相对于前面的 matchQuery()，这里提到的 multiMatchQuery("查询字符串","field1","field2",…)针对的是多个字段的搜索。也就是说，当 multiMatchQuery()中字段列表参数只有一个时，其作用与 matchQuery()相当；而当字段列表有 field1、field2 等多个参数时，则 field1 或者 field2 中包含指定的文本，都算检索到结果。代码段 4.23 给出了在 title 和 content 两个字段中对搜索词“中国”的检索。

代码段 4.23：使用 multiMatchQuery()实现跨字段检索

```
//import 语句,略
TransportClient client= new PreBuiltTransportClient(Settings.EMPTY)
    .addTransportAddress(new TransportAddress(InetAddress.getByName
    ("localhost"), 9300));
QueryBuilder qb= multiMatchQuery(
    "中国",                              //搜索词
    "title", "content"                  //要检索的字段
);
SearchResponse response= client.prepareSearch("it- home")
    .setTypes("_doc")
    .setSearchType(SearchType.DFS_QUERY_THEN_FETCH)
    .setQuery(qb)
    .setFrom(0)
    .setSize(10)
    .setExplain(true)
    .get();
for(SearchHit hit : response.getHits().getHits()) {
    System.out.println(hit.getSourceAsString());
}
client.close();
```

3. queryStringQuery()

queryStringQuery()查询支持 Lucene 所有的查询语法。对给定的内容，它会使用查询解析器来构造实际的查询，如 QueryBuilder qb＝QueryBuilders. queryStringQuery("＋ASP. NET -Java")，其含义是搜索包含 ASP. NET 且不含 Java 的结果集。相关实现见代码

段 4.24。

代码段 4.24：使用 queryStringQuery()方法执行查询

```
//import 语句,略
TransportClient client= new PreBuiltTransportClient(Settings.EMPTY)
      .addTransportAddress(new TransportAddress(InetAddress.getByName
      ("localhost"), 9300));
QueryBuilder qb= queryStringQuery("+ ASP.NET - Java");        //注意减号前有空格
SearchResponse response= client.prepareSearch("it- home")
      .setTypes("_doc")
      .setSearchType(SearchType.DFS_QUERY_THEN_FETCH)
      .setQuery(qb)
      .setFrom(0)
      .setSize(10)
      .setExplain(true)
      .get();
for(SearchHit hit : response.getHits().getHits()) {
      System.out.println(hit.getSourceAsString());
}
client.close();
```

4. simpleQueryStringQuery()

simpleQueryStringQuery()方法支持 Lucene 所有的查询语法。对于给定的内容，该查询使用解析器来构造实际的查询。查询过程中不会抛出任何异常，不可用的查询将自动被忽略。代码段 4.25 实现了对 title 字段中包含“程序员”和 Java 且 content 字段中不包含 ASP 的数据进行检索的功能，其中“title:程序员^2”是指在 title 字段中要包含“程序员”字符串且其权重为 2；“＋title:Java”是指在 title 字段中还要同时包含字符串 Java，但该字符串的权重为 1，这些权重会影响到最终结果排序；“-content:ASP”表示在 content 字段中不能有 ASP 字符串。

代码段 4.25：使用 simpleQueryStringQuery()实现类似 Lucene 的检索

```
//import 语句,略
TransportClient client= new PreBuiltTransportClient(Settings.EMPTY)
      .addTransportAddress(new TransportAddress(InetAddress.getByName
      ("localhost"), 9300));
QueryBuilder qb= simpleQueryStringQuery("title:程序员 ^2+ title:Java - content:
ASP");
SearchResponse response= client.prepareSearch("it- home")
```

```
        .setTypes("_doc")
        .setSearchType(SearchType.DFS_QUERY_THEN_FETCH)
        .setQuery(qb)
        .setFrom(0)
        .setSize(10)
        .setExplain(true)
        .get();
for(SearchHit hit : response.getHits().getHits()) {
    System.out.println(hit.getSourceAsString());
}
client.close();
```

4.4.7 词项检索

1. termQuery()

词项检索方法 termQuery("查询字段","查询词")的作用是在指定的字段中检索指定的查询词。例如 QueryBuilder qb=termQuery("content","程序员"),含义是在 content 字段中查询包括“程序员”字符串的结果集。针对 it-home 索引文件的相关实现如代码段 4.26 所示。

```
代码段 4.26: 使用 termQuery()方法查询词项
//import 语句,略
TransportClient client= new PreBuiltTransportClient(Settings.EMPTY)
        .addTransportAddress(new TransportAddress(InetAddress.getByName
        ("localhost"), 9300));
QueryBuilder qb= termQuery(
        "title",                                    //字段
        "程序员"                                    //查询词
);
SearchResponse response= client.prepareSearch("it-home")
        .setTypes("_doc")
        .setSearchType(SearchType.DFS_QUERY_THEN_FETCH)
        .setQuery(qb)
        .setFrom(0)
        .setSize(10)
        .setExplain(true)
        .get();
for(SearchHit hit : response.getHits().getHits()) {
    System.out.println(hit.getSourceAsString());
}
client.close();
```

2. termsQuery()

termsQuery("查询字段","field1","field2",…)方法允许在特定字段中匹配多个词项,检索某些同一个字段中包含多种不同信息的多个文档。例如,如果想查询在索引文件 it-home 中的 title 字段中包含字符串“程序员”或 Java 的文档,可以采用代码段 4.27 中的方法。

代码段 4.27: 使用 termsQuery()方法查询不同的词项

```
//import 语句,略
TransportClient client= new PreBuiltTransportClient(Settings.EMPTY)
     .addTransportAddress(new TransportAddress(InetAddress.getByName
     ("localhost"), 9300));
QueryBuilder qb= termsQuery(
     "title",                                    //字段
     "程序员", "Java"                             //多个查询词
);
SearchResponse response= client.prepareSearch("it-home")
     .setTypes("_doc")
     .setSearchType(SearchType.DFS_QUERY_THEN_FETCH)
     .setQuery(qb)
     .setFrom(0)
     .setSize(10)
     .setExplain(true)
     .get();
for(SearchHit hit : response.getHits().getHits()) {
     System.out.println(hit.getSourceAsString());
}
client.close();
```

3. rangeQuery()

rangeQuery("查询字段")方法是针对范围的查询,一般只作用在单个字段上,并且查询参数要封装在字段名称中,它也支持 from/to 或 gte/lte 等参数。代码段 4.28 实现了对发布时间在 2018-01-01 至今范围内数据的查询。类似地,可以使用其他时间表示方式,参见 3.3.2 节中关于 range 查询的部分。

代码段 4.28: 使用 rangeQuery()方法查询特定时间范围内的数据

```
//import 语句,略
TransportClient client= new PreBuiltTransportClient(Settings.EMPTY)
```

```
        .addTransportAddress(new TransportAddress(InetAddress.getByName
        ("localhost"), 9300));
QueryBuilder qb= rangeQuery("publishTime")          //指定查询范围的出版时间字段
        .gte("2018-01-01")                          //指定日期下限
        .lte("now");                                //指定日期上限
SearchResponse response= client.prepareSearch("it-home")
        .setTypes("_doc")
        .setSearchType(SearchType.DFS_QUERY_THEN_FETCH)
        .setQuery(qb)
        .setFrom(0)
        .setSize(10)
        .setExplain(true)
        .get();
for(SearchHit hit : response.getHits().getHits()) {
        System.out.println(hit.getSourceAsString());
}
client.close();
```

4. prefixQuery()

prefixQuery("查询字段"，"前缀")方法能够根据输入关键词开头的一部分来匹配整条数据。代码段 4.29 实现了对 title 字段中所有以“程序”开头的数据的检索。

代码段 4.29：使用 prefixQuery()方法查询指定前缀的数据

```
//import 语句,略
TransportClient client= new PreBuiltTransportClient(Settings.EMPTY)
        .addTransportAddress(new TransportAddress(InetAddress.getByName
        ("localhost"), 9300));
QueryBuilder qb= prefixQuery(
        "title",                            //字段
        "程序"                               //查询词前缀
);
SearchResponse response= client.prepareSearch("it-home")
        .setTypes("_doc")
        .setSearchType(SearchType.DFS_QUERY_THEN_FETCH)
        .setQuery(qb)
        .setFrom(0)
        .setSize(10)
        .setExplain(true)
        .get();
```

```
for(SearchHit hit : response.getHits().getHits()) {
    System.out.println(hit.getSourceAsString());
}
client.close();
```

5. wildcardQuery()

wildcardQuery("查询字段","含有通配符的查询词")方法的作用是在指定的字段中检索含有通配符的查询词，如 QueryBuilder qb＝wildcardQuery("content", "?国")。有关通配符检索的内容参见 3.3.2 节的叙述，这里不再赘述。在 it-home 索引文件中的相关实现见代码段 4.30。

```
代码段 4.30：使用 wildcardQuery()执行带有通配符的查询
//import 语句，略
TransportClient client= new PreBuiltTransportClient(Settings.EMPTY)
    .addTransportAddress(new TransportAddress(InetAddress.getByName
    ("localhost"), 9300));
QueryBuilder qb= wildcardQuery("title", "?国");                //字段和查询词
SearchResponse response= client.prepareSearch("it- home")
    .setTypes("_doc")
    .setSearchType(SearchType.DFS_QUERY_THEN_FETCH)
    .setQuery(qb)
    .setFrom(0)
    .setSize(10)
    .setExplain(true)
    .get();
for(SearchHit hit : response.getHits().getHits()) {
    System.out.println(hit.getSourceAsString());
}
client.close();
```

6. regexpQuery()

regexpQuery("查询字段" ,"正则表达式")方法的作用是利用正则表达式进行查询。文档中凡是正则表达式能够匹配的内容，其整条数据就会被作为查询结果返回。代码段 4.31 实现了对内容包含“Java 程序员”或 ASP.NET 的文档数据的查询。

```
代码段 4.31：使用正则表达式查询
//import 语句，略
TransportClient client= new PreBuiltTransportClient(Settings.EMPTY)
```

```
        .addTransportAddress(new TransportAddress(InetAddress.getByName
        ("localhost"), 9300));
QueryBuilder qb= regexpQuery(
        "content",                          //字段
        "[Java 程序员 | ASP.NET]"            //正则表达式
);
SearchResponse response= client.prepareSearch("it- home")
        .setTypes("_doc")
        .setSearchType(SearchType.DFS_QUERY_THEN_FETCH)
        .setQuery(qb)
        .setFrom(0)
        .setSize(10)
        .setExplain(true)
        .get();
for(SearchHit hit : response.getHits().getHits()) {
        System.out.println(hit.getSourceAsString());
}
client.close();
```

4.4.8 复合查询

1. boolQuery()

boolQuery()是一个由其他类型查询组合而成的文档匹配类型的查询，可由一个或者多个查询语句构成。查询语句中可以使用的匹配条件如下：

- boolQuery().must(termQuery("字段","查询内容"))：待匹配的文档的指定字段必须满足查询内容。
- boolQuery().should(termQuery("字段","查询内容"))：待匹配的文档的指定字段可以满足查询内容。
- boolQuery().mustNot(termQuery("字段","查询内容"))：待匹配的文档的指定字段必须不满足查询内容，但不能只用一个 mustNot 语句搜索文档。

上述匹配条件可以组合起来并作为 QueryBuilder 的具体方法。代码段 4.32 给出了一个形式化的例子。

代码段 4.32：boolQuery()查询的形式化表示

```
QueryBuilder qb= QueryBuilders.boolQuery()
        .must(termQuery("字段 1", "查询内容 1"))
```

```
        .must(termQuery("字段 2", "查询内容 2"))
        .mustNot(termQuery("字段 3", "查询内容 3"))
        .should(termQuery("字段 4", "查询内容 4"))
        .filter(termQuery("字段 5", "查询内容 5"));
```

代码段 4.33 给出了实际使用方法，这里是在索引文件 it-home 的 content 字段中查询包含“开发”和“算法”且不包含“教学”，同时在 content 字段中可以包含“代码”并且与 java 相关(但不影响分值)的结果集。

代码段 4.33：使用 boolQuery()执行查询

```
//import 语句，略
TransportClient client= new PreBuiltTransportClient(Settings.EMPTY)
        .addTransportAddress(new TransportAddress(InetAddress.getByName
        ("localhost"), 9300));
QueryBuilder qb= boolQuery()
        .must(termQuery("content", "开发"))
        .must(termQuery("content", "算法"))
        .mustNot(termQuery("content", "教学"))
        .should(termQuery("content", "代码"))
        .filter(termQuery("content", "java"));
SearchResponse response= client.prepareSearch("it-home")
        .setTypes("_doc")
        .setSearchType(SearchType.DFS_QUERY_THEN_FETCH)
        .setQuery(qb)
        .setFrom(0)
        .setSize(10)
        .setExplain(true)
        .get();
for(SearchHit hit : response.getHits().getHits()) {
        System.out.println(hit.getSourceAsString());
}
client.close();
```

2. boostingQuery()

boostingQuery()方法可将匹配的结果降级处理。与 bool 查询中的 not 子句不同，boostingQuery()查询结果中仍然会包含不符合预期的词项，但其分值会降低。代码段 4.34 实现了对标题含有 java 字符串的文档中内容与“求职”不相关的信息的查询。关键词“求职”处于后面的子句中，其结果的分值将会根据 negativeBoost()方法中指定的值降低。

```
代码段 4.34: 使用 boostingQuery()方法执行分值提升和降低的查询
//import 语句,略
TransportClient client= new PreBuiltTransportClient(Settings.EMPTY)
      .addTransportAddress(new TransportAddress(InetAddress.getByName
      ("localhost"), 9300));
QueryBuilder qb= boostingQuery(
      termQuery("category","java"),                    //要提升分值的字段
      termQuery("content","求职"))                     //要降低分值的字段
      .negativeBoost(0.2f);
SearchResponse response= client.prepareSearch("it- home")
      .setTypes("_doc")
      .setSearchType(SearchType.DFS_QUERY_THEN_FETCH)
      .setQuery(qb)
      .setFrom(0)
      .setSize(10)
      .setExplain(true)
      .get();
for(SearchHit hit : response.getHits().getHits()) {
      System.out.println(hit.getSourceAsString());
}
client.close();
```

4.4.9　跨度查询

1. spanTermQuery()

spanTermQuery()方法可以查询包含词项的一段文本。代码段 4.35 实现了对包含词项 java 的查询。

```
代码段 4.35: 使用 spanTermQuery()执行包含特定词项的文本的查询
//import 语句,略
TransportClient client= new PreBuiltTransportClient(Settings.EMPTY)
      .addTransportAddress(new TransportAddress(InetAddress.getByName
      ("localhost"), 9300));
QueryBuilder qb= spanTermQuery(
      "content",                                //字段
      "java"                                    //要查询的词项
);
SearchResponse response= client.prepareSearch("it- home")
```

```
.setTypes("posts")
        .setSearchType(SearchType.DFS_QUERY_THEN_FETCH)
        .setQuery(qb)
        .setFrom(0)
        .setSize(10)
        .setExplain(true)
        .get();
for(SearchHit hit : response.getHits().getHits()) {
        System.out.println(hit.getSourceAsString());
}
client.close();
```

2. spanOrQuery()

spanOrQuery()方法可以包含多个 spanTermQuery()方法，匹配其查询结果的并集。代码段 4.36 实现了对含有词项 java、json 或 jquery 的查询。

```
代码段 4.36: 使用 spanOrQuery()方法执行查询
//import 语句,略
TransportClient client= new PreBuiltTransportClient(Settings.EMPTY)
        .addTransportAddress(new TransportAddress(InetAddress.getByName
        ("localhost"), 9300));
QueryBuilder qb= spanOrQuery(spanTermQuery("content","java"))
        .addClause(spanTermQuery("content","json"))
        .addClause(spanTermQuery("content","jquery"));
SearchResponse response= client.prepareSearch("it-home")
        .setTypes("_doc")
        .setSearchType(SearchType.DFS_QUERY_THEN_FETCH)
        .setQuery(qb)
        .setFrom(0)
        .setSize(10)
        .setExplain(true)
        .get();
for(SearchHit hit : response.getHits().getHits()) {
        System.out.println(hit.getSourceAsString());
}
client.close();
```

3. spanContainingQuery()

spanContainingQuery()方法中包含 spanNearQuery()方法,能够通过两个词项确定一个文本跨度以及一个 spanTermQuery()查询,能够指定跨度中的第三个词项。在查询时如果发现前者的匹配项中包含了后者的匹配项,那么前者的匹配项将作为查询结果被返回。代码段 4.37 实现了在词项 windows 和 unix 的匹配项中对含有词项 mac 的匹配项的查询,spanNearQuery()方法中的数字为 slop 参数,指定了两个词项之间可以存在多少个不匹配的词项。

代码段 4.37: 使用 spanContainingQuery()方法执行查询

```
//import 语句,略
TransportClient client= new PreBuiltTransportClient(Settings.EMPTY)
      .addTransportAddress(new TransportAddress(InetAddress.getByName
      ("localhost"), 9300));
QueryBuilder qb= spanContainingQuery(
      spanNearQuery(spanTermQuery("content","windows"), 30)      //确定跨度的第一个词项及 slop 参数
      .addClause(spanTermQuery("content","unix"))       //确定跨度的第二个词项
      .inOrder(true),
      spanTermQuery("content","mac"));                  //两词项之间包含的词项
SearchResponse response= client.prepareSearch("it- home")
      .setTypes("_doc")
      .setSearchType(SearchType.DFS_QUERY_THEN_FETCH)
      .setQuery(qb)
      .setFrom(0)
      .setSize(10)
      .setExplain(true)
      .get();
for(SearchHit hit : response.getHits().getHits()) {
      System.out.println(hit.getSourceAsString());
}
client.close();
```

4. spanWithinQuery()

spanWithinQuery()方法中含有 spanNearQuery()方法,能够通过两个词项确定一个文本跨度以及一个 spanTermQuery()查询,能够指定跨度当中的第三个词项。在查询时如果发现前者的匹配项中包含了后者的匹配项,那么后者的匹配项将作为查询结果被返回。代码段 4.38 实现了在词项 windows 和 unix 的匹配项中对含有词项 mac 的匹配项的查询。

代码段 4.38：使用 spanWithinQuery()方法执行查询

```
//import 语句,略
TransportClient client= new PreBuiltTransportClient(Settings.EMPTY)
    .addTransportAddress(new TransportAddress(InetAddress.getByName
    ("localhost"), 9300));
QueryBuilder qb= spanWithinQuery(
    spanNearQuery(spanTermQuery("content","windows"), 30)    //确定跨度的第一个词项及 slop 参数
    .addClause(spanTermQuery("content","unix"))        //确定跨度的第二个词项
    .inOrder(true),
    spanTermQuery("content","mac"));                   //两词项之间包含的词项
SearchResponse response= client.prepareSearch("it- home")
    .setTypes("posts")
    .setSearchType(SearchType.DFS_QUERY_THEN_FETCH)
    .setQuery(qb)
    .setFrom(0)
    .setSize(10)
    .setExplain(true)
    .get();
for(SearchHit hit : response.getHits().getHits()) {
    System.out.println(hit.getSourceAsString());
}
client.close();
```

4.4.10 特殊查询

1. moreLikeThisQuery()

可以通过 moreLikeThisQuery()方法查询与指定文本相似的文档，实现方法见代码段 4.39。该代码段中定义了不同字段和查询文本的集合，作为参数传入 moreLikeThisQuery()方法中，并在其中使用 minTermFreq()方法和 maxQueryTerms()方法进行设置。其后的 minTermFreq()方法指定查询内容的最少出现次数，maxQueryTerms()方法指定显示匹配结果的最大条数。

代码段 4.39：使用 moreLikeThisQuery()执行查询

```
//import 语句,略
TransportClient client= new PreBuiltTransportClient(Settings.EMPTY)
    .addTransportAddress(new TransportAddress(InetAddress.getByName
    ("localhost"), 9300));
```

```
String[] fields= {"title", "content"};          //查询字段
String[] texts= {"可运行于多个平台"};            //指定要查询的内容
Item[] items= null;
QueryBuilder qb= moreLikeThisQuery(fields, texts, items)
    .minTermFreq(1)                             //指定最少出现多少次才能被检出
    .maxQueryTerms(12);                         //指定显示结果的最大条数
SearchResponse response= client.prepareSearch("it- home")
    .setTypes("posts")
    .setSearchType(SearchType.DFS_QUERY_THEN_FETCH)
    .setQuery(qb)
    .setFrom(0)
    .setSize(10)
    .setExplain(true)
    .get();
for(SearchHit hit : response.getHits().getHits()) {
    System.out.println(hit.getSourceAsString());
}
client.close();
```

Tips moreLikeThisQuery()方法根据字段、相似文本(likeTexts)、相似词项(likeItems)找到一个文档,再找到与这个文档相似的其他文档。其常用的 API 如下:

- moreLikeThisQuery(Item[] likeItems)
- moreLikeThisQuery(String[] fields, String[] likeTexts, Item[] likeItems)
- moreLikeThisQuery(String[] likeTexts)
- moreLikeThisQuery(String[] likeTexts, Item[] likeItems)

2. scriptQuery()

scriptQuery()方法可以通过使用 script 子句来定制表达式,执行查询时,script 子句可以生成一个由外部参数计算出的特定字段并插入到原来的查询语句中执行。script 子句默认使用的脚本语言是 painless。代码段 4.40 演示了对 replies 值大于 0 的文档的查询,在 scriptQuery()方法中传入的脚本信息为默认的 inline 格式。

代码段 4.40:使用 scriptQuery()执行查询

```
//import 语句,略
TransportClient client= new PreBuiltTransportClient(Settings.EMPTY)
    .addTransportAddress(new TransportAddress(InetAddress.getByName
    ("localhost"), 9300));
```

```
QueryBuilder qb= scriptQuery(
    new Script("doc['replies'].value> 0")        //直接传入脚本
);
SearchResponse response= client.prepareSearch("it- home")
    .setTypes("_doc")
    .setSearchType(SearchType.DFS_QUERY_THEN_FETCH)
    .setQuery(qb)
    .setFrom(0)
    .setSize(10)
    .setExplain(true)
    .get();
SearchHit[] hits= response.getHits().getHits();
for(SearchHit hit : hits) {
    System.out.println(hit.getSourceAsString());
}
client.close();
```

4.5 聚合

关于聚合，在 3.4 节中已给出了较为具体的描述。在 Java 接口的 API 中，聚合只有 metric 和 bucket 两种形式，并且是像查询一样添加到执行检索的代码中间执行的。此外，聚合 API 中还特别提供了专门用于接收聚合结果数据的方法。下面介绍两种聚合的编写方法，限于篇幅，本节只对部分聚合进行介绍，包括 metric 聚合中的 min 聚合、sum 聚合、stats 聚合、extended_stats 聚合、value_count 聚合，以及 bucket 聚合中的 terms 聚合、range 聚合、date_range 聚合、histogram 聚合、date_histogram 聚合等。

4.5.1 Metric 聚合

1. min 聚合、sum 聚合

在聚合中可以快捷地完成对最值、和、平均值等的统计。代码段 4.41 完成对指定字段的最小值聚合（若统计最大值，将代码段 4.41 中的 min 换成 max 即可，这里不再赘述）。其中，第一段是构造聚合的过程，执行后将生成该聚合的 RESTful 代码；第二段是执行数据检索的过程，会返回搜索任务的结果集；第三段是获取聚合统计结果的过程，统计值将被赋值给 double 型变量 value。

代码段 4.41：使用 min 聚合统计最小访问量

```
//import 语句,略
TransportClient client= new PreBuiltTransportClient(Settings.EMPTY)
    .addTransportAddress(new TransportAddress(InetAddress.getByName
    ("localhost"), 9300));
MinAggregationBuilder aggregation= AggregationBuilders
    .min("agg")                                          //聚合的名称
    .field("views");                                     //执行聚合的字段

SearchResponse sr= client.prepareSearch("it- home")      //创建 SearchResponse,指定索引文件
    .setTypes("_doc")                                    //指定类型文件
    .addAggregation(aggregation)                         //添加聚合
    .get();

Min agg= sr.getAggregations().get("agg");
double value= agg.getValue();                            //获取聚合统计结果
for(SearchHit hit : sr.getHits().getHits()) {
    System.out.println(hit.getSourceAsString());         //遍历检索结果
}
System.out.println("Minimum views is: "+ value);         //输出聚合统计结果
client.close();
```

sum 聚合只需在相应的类和方法的调用上稍加修改即可。代码段 4.42 完成对指定字段的求和(若统计平均值,将代码段 4.42 中的 sum 换成 avg 即可,不再赘述)。

代码段 4.42：使用 sum 聚合统计总访问量

```
//import 语句,略
TransportClient client= new PreBuiltTransportClient(Settings.EMPTY)
    .addTransportAddress(new TransportAddress(InetAddress.getByName
    ("localhost"), 9300));
SumAggregationBuilder aggregation= AggregationBuilders   //这里调用 SumAggregationBuilder 类
    .sum("agg")                                          //这里调用 sum()方法
    .field("views");
SearchResponse sr= client.prepareSearch("it- home")
    .setTypes("_doc")
    .addAggregation(aggregation)
```

```
        .get();
Sum agg= sr.getAggregations().get("agg");                    //这里调用 Sum类
double value= agg.getValue();
for(SearchHit hit : sr.getHits().getHits()) {
        System.out.println(hit.getSourceAsString());
}
System.out.println("Sum of views is: "+ value);
client.close();
```

2. stats 聚合、extended_stats 聚合

stats 聚合实现多值统计，返回值包括计数、最小值、最大值、平均值、求和等。代码段 4.43 演示了对相应字段进行多值统计的方法。

代码段 4.43：使用 stats 聚合进行多值统计

```
//import 语句，略
TransportClient client= new PreBuiltTransportClient(Settings.EMPTY)
        .addTransportAddress(new TransportAddress(InetAddress.getByName
        ("localhost"), 9300));
StatsAggregationBuilder aggregation= AggregationBuilders
        .stats("agg")
        .field("price");
SearchResponse sr= client.prepareSearch("it- home")
        .setTypes("_doc")
        .addAggregation(aggregation)
        .get();
Stats agg= sr.getAggregations().get("agg");
double min= agg.getMin();                   //求最小值
double max= agg.getMax();                   //求最大值
double avg= agg.getAvg();                   //求平均值
double sum= agg.getSum();                   //求和
long count= agg.getCount();                 //计数
for(SearchHit hit : sr.getHits().getHits()) {
        System.out.println(hit.getSourceAsString());
}
System.out.println(min+ "\t"+ max+ "\t"+ avg+ "\t"+ sum+ "\t"+ count);
client.close();
```

extended_stats 聚合则是对一般的 stats 聚合的功能扩展，可以在上述输出结果上添加平方和、方差和标准差等指标，参见代码段 4.44。

代码段 4.44：使用 extended_stats 聚合进行多值统计

```
//import 语句,略
TransportClient client= new PreBuiltTransportClient(Settings.EMPTY)
      .addTransportAddress(new TransportAddress(InetAddress.getByName
      ("localhost"), 9300));
ExtendedStatsAggregationBuilder aggregation= AggregationBuilders
      .extendedStats("agg")
      .field("views");
SearchResponse sr= client.prepareSearch("it- home")
      .setTypes("_doc")
      .addAggregation(aggregation)
      .get();
ExtendedStats agg= sr.getAggregations().get("agg");
double min= agg.getMin();                              //与上面的多值统计相同
double max= agg.getMax();
double avg= agg.getAvg();
double sum= agg.getSum();
long count= agg.getCount();
double stdDeviation= agg.getStdDeviation();            //标准差
double sumOfSquares= agg.getSumOfSquares();            //平方和
double variance= agg.getVariance();                    //方差
for(SearchHit hit : sr.getHits().getHits()) {
      System.out.println(hit.getSourceAsString());
}
System.out.println(min+ "\t"+ max+ "\t"+ avg+ "\t"+ sum+ "\t"+ count);
System.out.println(stdDeviation+ "\t"+ sumOfSquares+ "\t"+ variance);
client.close();
```

3. value_count 聚合

value_count 聚合是一种单值指标聚合，用来计算文档中某个字段的统计数据。代码段 4.45 实现对 views 字段执行 value_count 聚合。

代码段 4.45：使用 value_count 聚合进行计数

```
//import 语句,略
TransportClient client= new PreBuiltTransportClient(Settings.EMPTY)
      .addTransportAddress(new TransportAddress(InetAddress.getByName
      ("localhost"), 9300));
ValueCountAggregationBuilder aggregation= AggregationBuilders
      .count("agg")
```

```
        .field("views");               //对价格统计(每部手机对应一个价格)
SearchResponse sr= client.prepareSearch("it- home")
        .setTypes("_doc")
        .addAggregation(aggregation)
        .get();
ValueCount agg= sr.getAggregations().get("agg");
long value= agg.getValue();             //计数结果
for(SearchHit hit : sr.getHits().getHits()) {
        System.out.println(hit.getSourceAsString());
}
System.out.println("Total number is: "+ value);
client.close();
```

4.5.2 bucket 聚合

1. terms 聚合

terms 聚合用于对指定字段的内容进行分布统计。在聚合过程中会动态构建多个 bucket，并对每个 bucket 计算出一个特定的值。代码段 4.46 对 it-home 索引文件中的访问量分布情况进行了统计。

代码段 4.46：使用 terms 聚合统计访问量分布情况

```
//import 语句,略
TransportClient client= new PreBuiltTransportClient(Settings.EMPTY)
        .addTransportAddress(new TransportAddress(InetAddress.getByName
        ("localhost"), 9300));
SearchResponse sr= client.prepareSearch("it- home")
        .setTypes("_doc")
        .addAggregation(AggregationBuilders          //在执行检索的代码中定义聚合
            .terms("agg")                            //聚合的名称
            .field("views")                          //指定统计字段
        ).get();
Terms genders= sr.getAggregations().get("agg");      //获取统计结果
for(Terms.Bucket entry : genders.getBuckets()) {
        System.out.println(entry.getKeyAsString()+ ": "+ entry.getDocCount());
}
client.close();
```

2. range 聚合

range 聚合基于多个 bucket 完成指定范围内的统计，在每个 bucket 中定义一个范围，用于统计指定字段在该范围内的值。在聚合过程中，从每个文档中提取的值将针对指定的范围进行检查，并且返回相关或匹配的文档。代码段 4.47 实现了对 it-home 索引中 views 字段的值分别在 3 个范围内的数量统计。

代码段 4.47：使用 range 聚合进行指定范围的统计

```
//import 语句，略
//主类定义，略
private static final Logger logger= (Logger) LogManager.getLogger(ClientTest.class);     //用 log4j2
记录日志
//main()主方法定义，略
TransportClient client= new PreBuiltTransportClient(Settings.EMPTY)
      .addTransportAddress(new TransportAddress(InetAddress.getByName
      ("localhost"), 9300));
AggregationBuilder aggregation= AggregationBuilders
      .range("agg")
      .field("views")
      .addUnboundedTo(100.0f)                        //指定 0~100 的范围
      .addRange(200.0f, 400.0f)                      //指定 200~400 的范围
      .addUnboundedFrom(500.0f);                     //指定 500 以上的范围
SearchResponse sr= client.prepareSearch("it- home")
      .setTypes("_doc")
      .addAggregation(aggregation)
      .get();
Range agg= sr.getAggregations().get("agg");
for(Range.Bucket entry : agg.getBuckets()) {
      String key= entry.getKeyAsString();            //范围的类别
System.out.println(key);
      Number from= (Number) entry.getFrom();         //范围的下界
System.out.println(from);
      Number to= (Number) entry.getTo();             //范围的上界
System.out.println(to);
      long docCount= entry.getDocCount();            //计数
System.out.println(docCount);
      logger.info("key [{}], from [{}], to [{}], doc_count [{}]", key, from, to, docCount);     //记录日志
}
client.close();client.close();
```

3. date_range 聚合

date_range 聚合是专门对时间类型的字段进行区段统计的聚合。代码段 4.48 介绍了其基本使用方法。

代码段 4.48：使用 date_range 聚合进行多值统计

```
//import 语句,略
//主类定义,略
private static final Logger logger= (Logger) LogManager.getLogger(ClientTest.class);     //用 log4j2
记录日志
//main()主方法定义,略
TransportClient client= new PreBuiltTransportClient(Settings.EMPTY)
      .addTransportAddress(new TransportAddress(InetAddress.getByName
      ("localhost"), 9300));
AggregationBuilder aggregation= AggregationBuilders
      .dateRange("agg")
      .field("publishTime")
      .format("yyyy-MM-dd HH:mm:ss")
      .addRange("2018-01-01 00:00:00", "2018-07-01 00:00:00");     //指定时间范围
SearchResponse sr= client.prepareSearch("it-home")
.setTypes("_doc")
.addAggregation(aggregation)
.get();
Range agg= sr.getAggregations().get("agg");
for (Range.Bucket entry : agg.getBuckets()) {
String key= entry.getKeyAsString();                          //时间区段
      DateTime fromAsDate= (DateTime) entry.getFrom();         //起始时间
      DateTime toAsDate= (DateTime) entry.getTo();             //截止时间
      long docCount= entry.getDocCount();                      //计数
      logger.info("key [{}], from [{}], to [{}], doc_count [{}]", key, fromAsDate, toAsDate, docCount);
}
client.close();
```

4. histogram 聚合

利用 histogram 聚合可以根据返回值(针对数值型或日期型的字段)生成可创建柱状图的聚合数据。代码段 4.49 是计算 views 字段以 100 为步长的各区段的统计分布情况。

代码段 4.49: 使用 histogram 聚合进行柱状图统计

```
//import 语句,略
//主类定义,略
private static final Logger logger= (Logger) LogManager.getLogger(ClientTest.class);
                                                                  //用 log4j2 记录日志
//main()主方法定义,略
TransportClient client= new PreBuiltTransportClient(Settings.EMPTY)
    .addTransportAddress(new TransportAddress(InetAddress.getByName
    ("localhost"), 9300));
AggregationBuilder aggregation= AggregationBuilders
    .histogram("agg")
    .field("views")
    .interval(100);                              //步长为 100
SearchResponse sr= client.prepareSearch("it-home")
    .setTypes("_doc")
    .addAggregation(aggregation)
    .get();
Histogram agg= sr.getAggregations().get("agg");
for(Histogram.Bucket entry : agg.getBuckets()) {
    Number key= (Number) entry.getKey();
    System.out.println(key);
    long docCount= entry.getDocCount();
    System.out.println(docCount);
    logger.info("key [{}], doc_count [{}]", key, docCount);
}
client.close();
```

5. date_histogram 聚合

date_histogram 聚合是一个增强型的专门用于日期型字段统计的 histogram 聚合,它允许使用 year、month、week、day、hour、minute 等常量作为 interval 属性的取值。代码段 4.50 实现了以下聚合操作:在 field 中填写一个日期类型的字段名称,在 interval 中填写一个步长值,通过 format 参数设置时间格式。

代码段 4.50: 使用 date_histogram 聚合执行不同时间范围的统计

```
//import 语句,略
//主类定义,略
private static final Logger logger= (Logger) LogManager.getLogger(ClientTest.class);
```

```
//main()主方法定义,略
TransportClient client= new PreBuiltTransportClient(Settings.EMPTY)
        .addTransportAddress(new TransportAddress(InetAddress.getByName
        ("localhost"), 9300));
AggregationBuilder aggregation= AggregationBuilders
        .dateHistogram("agg")
        .field("publishTime")
        .dateHistogramInterval(DateHistogramInterval.hours(3));    //步长为 3h
SearchResponse sr= client.prepareSearch("it- home")
        .setTypes("_doc")
        .addAggregation(aggregation)
        .get();
Histogram agg= sr.getAggregations().get("agg");
for(Histogram.Bucket entry : agg.getBuckets()) {
        DateTime key= (DateTime) entry.getKey();
        String keyAsString= entry.getKeyAsString();
        long docCount= entry.getDocCount();
        logger.info("key [{}], date [{}], doc_count [{}]", keyAsString,
        key.getYear(), docCount);
}
client.close();
```

4.6 对检索结果的进一步处理

4.6.1 控制每页的显示数量及排序依据

在上述例子中多次出现 setFrom()、setSize()、setExplain()子句。这是在搜索结果(例如 prepareSearch 中)通过 setFrom(起始检索结果)和 setSize(每页显示结果集)等参数来控制显示数量的一种方案,可以实现对检索结果的分页。同时,可以使用 setExplain(true)参数来设置是否需要对文档的打分情况进行解释。示例程序如图 4.9 所示。在图 4.9 下部的控制台中可以看出,检索结果正常,并给出了文档的排序依据,而这正是 setExplain(true)参数的作用。

4.6.2 基于 scroll 分页显示检索结果

既然能控制检索结果的显示数量,就能对检索结果进行翻页处理。scroll 不用于实时请求,而是用于处理大量类似数据,可以用一个语句检索大量甚至所有的数据,这与传统的

```
QueryBuilder qb = termQuery(
        name: "content",
        value: "java"
);
SearchResponse response = client.prepareSearch( ...indices: "it-home")
        .setTypes("_doc")
        .setSearchType(SearchType.DFS_QUERY_THEN_FETCH)
        .setQuery(qb)
        .setFrom(0)
        .setSize(20)
        .setExplain(true)
        .get();
System.out.println(response.status());
System.out.println();
System.out.println(response.getHits().getAt( position: 0).getExplanation());
client.close();
```

```
Run  ClientTest
/usr/jdk1.8.0_112/bin/java ...
OK

1.9626691 = weight(content:java in 41) [PerFieldSimilarity], result of:
  1.9626691 = score(doc=41,freq=109.0 = termFreq=109.0
), product of:
    0.9035754 = idf, computed as log(1 + (docCount - docFreq + 0.5) / (docFreq
      324.0 = docFreq
      800.0 = docCount
    2.1721144 = tfNorm, computed as (freq * (k1 + 1)) / (freq + k1 * (1 - b + b
      109.0 = termFreq=109.0
      1.2 = parameter k1
      0.75 = parameter b
      1067.5475 = avgFieldLength
      1304.0 = fieldLength


Process finished with exit code 0
```

图 4.9 检索数据并返回排序依据

使用游标查询关系数据库的方法类似。代码段 4.51 演示了基于 scroll 分页显示检索结果功能的实现，其中指定了检索字段和关键词。

代码段 4.51：基于 scroll 分页显示检索结果

```
//import 语句,略
TransportClient client= new PreBuiltTransportClient(Settings.EMPTY)
      .addTransportAddress(new TransportAddress(InetAddress.getByName
      ("localhost"), 9300));
QueryBuilder qb= termQuery("content", "java");          //指定字段和关键词
SearchResponse scrollResp= client.prepareSearch()
      .addSort(FieldSortBuilder.DOC_FIELD_NAME, SortOrder.ASC)          //设置排序方式
      .setScroll(new TimeValue(60000))          //设置上下文过期时间(ms)
      .setQuery(qb)
      .setSize(10).get();          //设置每页显示的结果数量
do {
for(SearchHit hit : scrollResp.getHits().getHits()) {
      System.out.println(hit.getSourceAsString());
      }
```

```
        scrollResp= client.prepareSearchScroll(scrollResp.getScrollId()).
            setScroll(new TimeValue(60000)).execute().actionGet();
    System.out.println("===================================");
                                                        //每页显示结果的分隔线
  } while (scrollResp.getHits().getHits().length != 0);     //如果后面没有结果则停止
  client.close();
```

在代码 4.51 中，首先在 termQuery()方法中设置检索字段和关键词两个参数，将其赋值给 QueryBuilder 类的静态实例 qb；在使用 scroll 时，在 client. prepareSearch()方法中不指定索引文件和类型文件名称，在 setQuery()方法中传入查询对象实例 qb；setScroll()方法中的参数是过期时间，程序中设置的是 1min，1min 之后该上下文就不存在了，需重新构建 context，也就是说，在 setScroll()中设置每一个 scroll 上下文的存活时间；setSize()方法控制每页显示结果的数量。然后通过循环来遍历结果集中的数据，具体实现是：在 for 循环中使用 response. getHits(). getHits()方法遍历检索结果集，在循环体中使用 getSourceAsString()将每条检索结果显示出来。在 for 循环之后，使用 scrollResp. getScrollId()来获取此次检索结果的 scrollID。此时循环中的 client. prepareSearchScroll()方法可以利用获取的 scrollID 进行二次搜索，并返回结果。代码段 4.51 的执行结果如图 4.10 所示。

```
"title":"详解java面试题中的i++和++i","url":"http://bbs.it-home.org/thread-103273-1-5.html","con
"title":"jsp页面中获取servlet请求中的参数的办法详解","url":"http://bbs.it-home.org/thread-103280-1
"title":"C#中的函数式编程：序言（一）","url":"http://bbs.it-home.org/thread-102323-1-2.html","con
"title":"C# 程序之间传参数，Args 接收参数的处理","url":"http://bbs.it-home.org/thread-102327-1-2.h
"title":"C#服务器端生成报告文档：使用帆软报表生成Word、Pdf报告","url":"http://bbs.it-home.org/thread-
"title":"基于Json序列化和反序列化通用的封装完整代码","url":"http://bbs.it-home.org/thread-100722-1-
"title":"C#泛型类创建与使用的方法","url":"http://bbs.it-home.org/thread-100714-1-3.html","conten
title":"C# NetCDF文件64位读取","url":"http://bbs.it-home.org/thread-101619-1-2.html","content"
title":"HttpHelper类的调用方法详解","url":"http://bbs.it-home.org/thread-100673-1-3.html","cont
title":"C#使用Redis的基本操作","url":"http://bbs.it-home.org/thread-100686-1-3.html","content"

title":"asp.net MVC下使用rest的方法","url":"http://bbs.it-home.org/thread-98103-1-4.html","con
"title":"java 单例模式（饿汉模式与懒汉模式）","url":"http://bbs.it-home.org/thread-98106-1-4.html"
title":"微信JS-SDK分享功能的.Net实现代码","url":"http://bbs.it-home.org/thread-98104-1-4.html","
title":"C#事件实例详解","url":"http://bbs.it-home.org/thread-100622-1-4.html","content":"C#事件
```

图 4.10　分页显示检索结果

4.7　Java High Level RESTful Client 和 Elasticsearch DSL

4.7.1　Java High Level RESTful Client

使用 Java 以 RESTful 风格向 Elasticsearch 提交请求，是 Elasticsearch 6.0 开始加入的新特性。Java High Level RESTful Client 比 TransportClient 方式更简单易用，以传参的形式接收请求，与 Elasticsearch 交互并返回响应结果。该组件依赖于 Elasticsearch，在应用中

可以接收与 TransportClient 相同的参数，并返回与之相同的响应信息。

调用 Java High Level RESTful Client 需要加载特定的 JAR 包，然而这些 JAR 包并不在 Elasticsearch 的安装目录下，此时可以通过 Maven 来自动获取。代码段 4.52 展示了在 pom.xml 配置文件中写入的配置信息。其中的 6.2.4 为本例使用的 Elasticsearch 版本，读者可以根据实际情况进行修改。

代码段 4.52：在 pom.xml 配置文件中写入的配置信息

```
<dependency>
    <groupId>org.elasticsearch.client</groupId>
    <artifactId>elasticsearch-rest-high-level-client</artifactId>
    <version>6.2.4</version>
</dependency>
```

可通过手动加载 JAR 包的形式来导入依赖，即在 http://maven.aliyun.com 中获取 elasticsearch-rest-client-6.2.4.jar 包和 elasticsearch-rest-high-level-client-6.2.4.jar 包，然后将其导入开发平台的 library 中。

下面针对 Elasticsearch 索引和搜索等方面的功能，对 Java High Level RESTful Client 的应用进行介绍。使用该组件创建新索引文件 example 的示例代码如代码段 4.53 所示，其中包含了创建 HTTP 连接对象、创建索引、设置索引分片和副本、为索引设置映像、执行并查看执行结果等功能。

代码段 4.53：创建索引文件

```
//import 语句,略
RestHighLevelClient client=new RestHighLevelClient(
        RestClient.builder(new HttpHost("localhost", 9200, "http")));
                                                    //这里连接 HTTP 的 9200 端口
CreateIndexRequest request=new CreateIndexRequest("example");
                                                    //创建名为 example 的索引文件
    request.settings(Settings.builder()
        .put("index.number_of_shards", 2)           //设置分片数量
        .put("index.number_of_replicas", 2)         //设置副本数量
);
request.mapping("_doc",                             //设置映像,参数为 JSON 格式
        "  {\n"+
        "    \"_doc\": {\n"+
        "      \"properties\": {\n"+
```

```
        "            \"content\": {\n"+
        "                \"type\": \"text\"\n"+
        "            }\n"+
        "        }\n"+
        "    }\n"+
        "  }",
XContentType.JSON);
CreateIndexResponse createIndexResponse= client.indices().create(request);        //执行创建
boolean acknowledged= createIndexResponse.isAcknowledged();                       //查看执行结果
System.out.print("acknowledged: "+ acknowledged+ ", ");
boolean shardsAcknowledged= createIndexResponse.isShardsAcknowledged();
System.out.println("shardsAcknowledged: "+ shardsAcknowledged);
client.close();
```

在已有的索引中，按特定 id 号查询数据的操作如代码段 4.54 所示。其中 GetRequest 类的实例需要传入 3 个参数：索引文件名、统一的类型文件名和文档的 id 号。后面获取查询结果的代码与 TransportClient 相似。

代码段 4.54：按文档的 id 号查询数据

```
//import 语句，略
RestHighLevelClient client= new RestHighLevelClient(
        RestClient.builder(new HttpHost("localhost", 9200, "http")));
GetRequest getRequest= new GetRequest(
        "it-home",                          //要查询的索引文件名
        "_doc",                             //统一的类型文件名
        "6C8sxmMBmDMkTsldTkEC");            //要查询的 id 号
GetResponse getResponse= client.get(getRequest);
if(getResponse.isExists()) {
    Map<String, Object> sourceAsMap= getResponse.getSourceAsMap();      //获取查询结果的哈希表
    String title= sourceAsMap.get("title").toString();                  //获取查询结果数据
    String editorName= sourceAsMap.get("editorName").toString();
    String publishTime= sourceAsMap.get("publishTime").toString();
    System.out.println(title+ "\n"+ editorName+ "\n"+ publishTime);
} else {
    System.out.println("查询操作发生错误。");
}
client.close();
```

下面给出使用 query DSL 执行全文检索的示例代码。在索引文件 it-home 中搜索标题

中含有“程序员”字符串的全文检索程序如代码段 4.55 所示，该段代码由创建搜索、指定搜索详细配置信息、设置高亮字段、执行搜索和获取高亮字段搜索结果信息等部分构成。

代码段 4.55：搜索标题中含有“程序员”字符串的文档

```
//import 语句,略
RestHighLevelClient client= new RestHighLevelClient(
        RestClient.builder(new HttpHost("localhost", 9200, "http")));
SearchRequest searchRequest= new SearchRequest("it-home");            //指定索引文件名
SearchSourceBuilder sourceBuilder= new SearchSourceBuilder();         //开始构建搜索
sourceBuilder.query(QueryBuilders.matchQuery("title", "程序员"));
                                                            //指定全文检索的内容
sourceBuilder.from(0);                                      //指定查询结果的起始位置
sourceBuilder.size(5);                                      //指定查询结果的数量
sourceBuilder.sort(new FieldSortBuilder("views").order(SortOrder.ASC));
                                                            //指定排序方式
String[] includeFields= new String[] {"title", "editorName", "publishTime"};
                                                            //指定结果中要保留的字段
String[] excludeFields= new String[] {"_type"};             //指定结果中要排除的字段
sourceBuilder.fetchSource(includeFields, excludeFields);
HighlightBuilder highlightBuilder= new HighlightBuilder();
HighlightBuilder.Field highlightTitle= new HighlightBuilder.Field("title");
                                                            //指定高亮字段
highlightTitle.highlighterType("unified");
highlightBuilder.field(highlightTitle);
HighlightBuilder.Field highlightEditor= new HighlightBuilder.Field("editorName");
highlightBuilder.field(highlightEditor);
sourceBuilder.highlighter(highlightBuilder);
SearchSourceBuilder searchSourceBuilder= new SearchSourceBuilder();
searchRequest.source(sourceBuilder);
SearchResponse searchResponse= client.search(searchRequest);          //开始搜索
SearchHits hits= searchResponse.getHits();                  //处理搜索结果
SearchHit[] searchHits= hits.getHits();
for (SearchHit hit : searchHits) {
    Map<String, HighlightField> highlightFields= hit.getHighlightFields();
                                                            //获取高亮字段的搜索结果
    HighlightField highlight= highlightFields.get("title");
    Text[] fragments= highlight.fragments();
    String fragmentString= fragments[0].string();
```

```
        System.out.println(fragmentString);
    }
    client.close();
```

4.7.2 Elasticsearch DSL

Elasticsearch DSL 是 Elastic 公司提供的使用 Python 实现的 High Level 库，目的是方便编写和运行 Elasticsearch 的查询语句。Elasticsearch DSL 的安装比较简单，这里给出基于 pip 的安装命令：

```
pip install elasticsearch-dsl
```

安装成功后，便可进入 Python 环境，导入 Elasticsearch DSL。下面对 Elasticsearch DSL 提供的 API 给出一个查询和聚合实例。

代码段 4.56 首先创建客户端，创建 match 查询语句，查询条件为 os 是 Window 7 的日志，然后在查询结果集中进行聚合，找出日志长度最小值。

代码段 4.56：查询 os 为 Windows 7 的请求日志，并在结果集中找出日志长度最小值

```
from elasticsearch_dsl import Search
from elasticsearch import Elasticsearch
es_client=Elasticsearch()
search=Search(using=es_client, index='whale', doc_type='log')\                //创建查询语句
        .query("match", os="Windows 7")
search.aggs.bucket('min_size', 'min', field="log_size")
print(search.to_dict())              //将查询对象转为字典并输出
response=search.execute()            //执行搜索语句
for hit in response:
    print(hit.log_size, hit.os)
for tag in response.aggs:
    print(tag)
```

在图 4.11 中，第一行是查询语句，可参照第 3 章的内容进行理解；中间部分是输出的查询结果；最后一行输出对查询结果集进行聚合的结果。

通常在开发应用程序时会首先创建数据模型，然后可对数据进行增、删、查、改操作。在 Elasticsearch DSL 中提供了类似的方法，允许用户定义数据模型并与索引文件中的字段一一映射。代码段 4.57 首先创建了索引文件 whale 的数据模型类(设置的字段应与索引文件

```
查询语句：{'query': {'match': {'os': 'Windows 7'}}, 'aggs': {'min_size': {'min': {'field': 'log_size'}}}}

查询结果：
log size:204 os:Windows 7
log size:54 os:Windows 7
log size:462 os:Windows 7
log size:262 os:Windows 7
log size:329 os:Windows 7
log size:317 os:Windows 7
log size:394 os:Windows 7
log size:75 os:Windows 7
log size:481 os:Windows 7
log size:146 os:Windows 7

日志长度最小值：
{'value': 54.0}
```

图 4.11　查询输出内容

中的字段保持一致），使用 init()方法可以设置 whale 的映像，用简洁的方式实现了对 id 号为 1 的文档数据的操作。

代码段 4.57：利用数据模型对数据进行操作

```
from elasticsearch_dsl import DocType, Long, Keyword
from elasticsearch_dsl.connections import connections
connections.create_connection(hosts=['localhost'])                    //创建连接
class Whale(DocType):          //创建索引文件 whale 的数据模型类
    http_method=Keyword()
    status_code=Long()
    os=Keyword()
    log_size=Long()
    custom_ip=Keyword()
    uri=Keyword()
    timestamp=Keyword()
    class Meta:                //设置元信息，可设置索引文件名和类型文件名
        index='whale'
        doc_type='log'
Whale.init()                   //必须执行 init()方法初始化 whale 的映像
whale=Whale(meta={'id': 1},
            http_method='GET',
            status_code='200',
            os="Ubuntu 16.04",
            log_size=129,
            custom_ip="192.168.0.29",
            uri="/test.html",
```

```
                timestamp="08/11/2018 11:08:00")
whale.save()//保存上一步创建的日志信息
print(whale.to_dict())
whale=Whale.get(id="1")//获取 id号为 1的文档数据
print(whale.to_dict())

whale.update(http_method='POST', status_code=404)          //更新 id号为 1的文档的部分数据
print(Whale.get(id="1").to_dict())
```

Tips Whale 模型数据类中的 timestamp 字段本应该是 Date 类型，但由于 Elasticsearch DSL 6.1 未实现在请求数据时对 Date 类型的字段进行格式化操作的功能，所以，在请求时，该字段会被格式化为 Elasticsearch 默认的日期时间格式，这会与之前在 Elasticsearch 中设置的日期时间格式发生冲突，导致添加失败。因而，代码段 4.57 将 timestamp 设置为 Keyword 类型，保证其日期时间格式在请求过程中不会被修改，从而实现了添加数据的功能。

4.8 实例

下面介绍一个基于网络爬虫 WebMagic(它是一个无须配置、便于二次开发的爬虫框架)采集新闻数据，并利用 Elasticsearch 完成信息索引、检索的实例，它主要分 4 个部分：

(1) 使用 WebMagic 定制爬虫，抓取新闻数据。

(2) 在对获取的新闻数据进行一定的处理后存入 Elasticsearch，并对新闻数据建立倒排索引。

(3) 根据用户在前台页面输入的查询关键字，在后台的 Elasticsearch 索引库的指定字段完成查询。

(4) 传送用户检索的新闻数据以及推荐的相关数据到前台。

限于篇幅，这里主要介绍在 Elasticsearch 中的索引、检索等内容的实现。

4.8.1 在 Elasticsearch 中建立索引

Elasticsearch 规定，一般情况下，索引文件被创建好之后是不允许修改的。为了确保索引文件及类型文件中映像的正确无误，需要在使用爬虫获取新闻数据之前事先为 Elasticsearch 建立一个映像。代码段 4.58 演示了为 ifeng 索引文件手动配置映像的代码。

该段代码可以直接放入 head 工具的复合查询窗口中执行，在索引文件的位置只填写 ifeng 即可。

代码段 4.58：为 ifeng 索引文件配置映像

```
curl -XPUT localhost:9200/ifeng -d '{
  "mappings": {
    "_doc": {
      "properties": {
        "url": {
          "type": "keyword"
        },
        "title": {
          "type": "text",
          "index": true,
          "analyzer": "ik_max_word"
        },
        "publishTime": {
          "type": "date",
          "format": "yyyy年 MM月 dd日 HH:mm:ss||yyyy年 MM月 dd日"
        },
        "content": {
          "type": "text",
          "index": true,
          "analyzer": "ik_max_word"
        }
      }
    }
  }
}'
```

4.8.2 连接 Elasticsearch

新建一个名称为 ESClientHelper 的类，目的是完成 Elasticsearch 的客户端实例化工作。这里采用基于 TransportClient 的方式，可以通过调用 addTransportAddress()方法添加节点，如代码段 4.59 所示。其中 client 实例的初始化代码写在了一个 static 块中，实际运行时，该实例将常驻内存，以提高 Elasticsearch 客户端的响应速度。

代码段 4.59：基于 TransportClient 的方式初始化客户端

```
//import 语句,略
public class ESClientHelper {
    public static TransportClient client;
    static {
        try {
            client= new PreBuiltTransportClient(Settings.EMPTY)
                .addTransportAddress(
                  new TransportAddress(InetAddress.getByName("localhost"), 9300));
        } catch (UnknownHostException e) {
            e.printStackTrace();
        }
    }
}
```

4.8.3 信息采集与索引构建

WebMagic 模块分为 Spider、Downloader、PageProcessor、Scheduler、Pipeline 等多个部分。用户可以通过定义相关模块的接口，并将其不同的实现注入主框架类 Spider 来实现相关的信息采集功能。其结构如图 4.12 所示。其中，Spider 类是爬虫框架的核心组件，其接口调用采用了链式的 API 设计，其他功能全部通过接口注入 Spider 实现；Downloader 负责页面下载（使用 HttpClientDownloader 下载网页）；PageProcessor 负责页面分析及链接抽取，这里说的页面分析主要指对 HTML 页面的分析，通过编写一个实现 PageProcessor 接口的类可定制爬虫，而页面抽取最基本的方式是使用 XPath；Scheduler 负责 URL 去重等管理工作；Pipeline 可实现抽取结果的持久化（例如序列化到数据库中，甚至将其保存到基于键-值对的缓存中）。

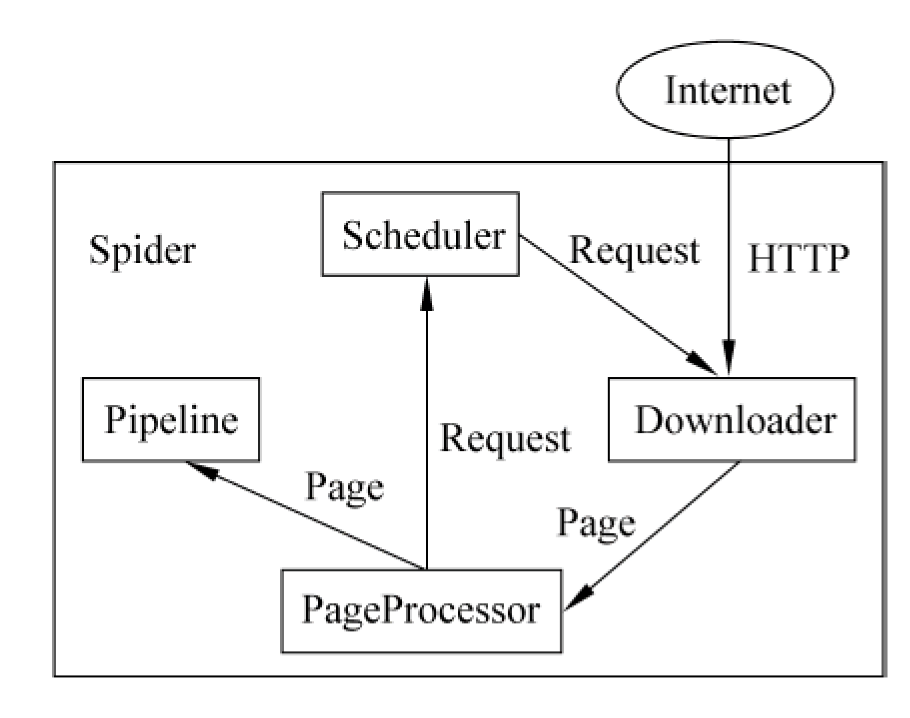


图 4.12 WebMagic 模块的结构

在构建完爬虫后，已实现了从 Internet 获取信息的目的，但它们需要持久化到索引文件中。在 4.1 节中已完成 Elasticsearch 的客户端实例化工作（让抓取的数据存储在名为 news 的索引库中）。这里为表述方便，在 head 工具中手动创建名为 news 的索引库。在本索引库中需要用到的字段和说明如表 4.1 所示。

表 4.1 news 索引库主要字段说明

字段名	作用及描述	字段名	作用及描述
id	文档编号，由 Elasticsearch 自动生成	content	抓取新闻正文内容
title	抓取的新闻标题	url	抓取的新闻页面 URL
publishTime	抓取的新闻发布时间		

下一步工作是在浏览器中使用开发者工具解析网页，实现 WebMagic 的 PageProcessor 接口并重写 process()方法，再配合 XPath 解析式即可获取相应的新闻标题、URL、正文及标签等，其核心实现如代码段 4.60 所示，其中的 page 对象是重写 process()方法时传入的参数。WebMagic 已经写好抽取网页内容的 Page 类，直接实例化作为参数即可。getHtml()以及下面的 getUrl()和 smartContent()是 WebMagic 中已经写好的方法，直接调用即可。

代码段 4.60：实现 PageProcessor 接口，解析新闻网页

```
//解析网页得到的新闻 URL
String url=page.getUrl().get();
//解析网页得到的新闻标题
String title=null==page.getHtml().xpath("//h1[@ id='artical_topic']/
text()").toString() ?
    page.getHtml().xpath("//h1/text()").toString() :
    page.getHtml().xpath("//h1[@ id='artical_topic']/text()").toString();
//解析网页得到的新闻发布时间
String publishTime=null==page.getHtml().xpath("//p[@ class='p_time']/span/text()").toString() ?
    page.getHtml().xpath("//div[@ class='hdpTit clearfix']/div[1]/p/span/text()").toString() :
    page.getHtml().xpath("//p[@ class='p_time']/span/text()").toString();
//使用框架中的功能自动获取新闻正文内容
String content=page.getHtml().smartContent().toString();
```

考虑到处理速度，可使用 JSON 数据交换格式。在上面的工作中已经抓取了新闻的标题、正文以及标签。在这一步首先调用 jsonBuilder()方法把获取的结果集转换为 JSON 格式，然后使用 Gson 类下的 toJson()方法传入相应参数，完成把抓取的数据转换为 JSON 格式并写进数据库的工作。具体方法如代码段 4.61 所示，此例中使用 prepareIndex()方法建立索引，并在 setSource()方法中利用第二个参数指定了输入数据的格式为 JSON。

代码段 4.61：将抓取到的数据持久化到 Elasticsearch 的索引库中

```
//import 语句，略
ESClient esClient=new ESClient();
Client client=esClient.getclient();          //得到与 Elasticsearch 链接的实例
//解析网页的代码，略
Map<String, Object>map=new HashMap<>();
map.put("url", url);
map.put("title", title);
map.put("publishTime", publishTime);
map.put("content", content);
String s=new Gson().toJson(map);             //把获取的网页内容转换为 JSON 格式
IndexResponse response=client.prepareIndex("ifeng", "_doc").setSource(s, XContentType.JSON).get();
//把转换后的 JSON 数据存入 ifeng 索引库中
```

程序执行后返回 head 并刷新，发现库中已添加了 publishTime、title、url、content 等需要抓取的数据列名，这相当于传统数据库中的各个字段。各个列下存储的就是抓取的信息，如图 4.13 所示。

查询 5 个分片中用的 5 个. 942 命中. 耗时 0.010 秒

_index	_type	_id	_score ▲	publishTime	title	url
ifeng	_doc	pZ7N0GMBthV094w7bC34	1	2018年06月04日 09:01:31	1年7次约会4进酒店！赵丽颖冯绍峰发的糖都要蛀牙了	http://ent.ifeng.com/a/20180
ifeng	_doc	v57N0GMBthV094w7iC3u	1	2018年06月05日 16:15:44	张一山也救不了新《家有儿女》	http://ent.ifeng.com/a/20180
ifeng	_doc	wJ7N0GMBthV094w7iS0e	1	2018年06月05日 13:51:09	《101》"隐藏版淘汰规定"：禁用手机 慎与异性接触	http://ent.ifeng.com/a/20180
ifeng	_doc	qp7N0GMBthV094w7ci0Q	1	2018年06月04日 07:50:38	明星收入究竟有多高，都有哪些避税手段？	http://ent.ifeng.com/a/20180
ifeng	_doc	0Z7N0GMBthV094w7nS1Z	1	2018年06月05日 16:19:06	夸海口要送豪宅给太太，婚姻不稳改送自己老妈了…	http://ent.ifeng.com/a/20180
ifeng	_doc	3Z7N0GMBthV094w7sC3o	1	2018年06月05日 11:19:23	《天下第一镖局》曝先导海报预告 功夫高手联袂打造	http://ent.ifeng.com/a/20180
ifeng	_doc	op7N0GMBthV094w7aC1b	1	2018年06月04日 15:00:48	郭敬明一身名牌霸气亮相机场 网友：穿出了童装味道	http://ent.ifeng.com/a/20180
ifeng	_doc	057N0GMBthV094w7oS0-	1	2018年06月05日 11:11:53	《家有儿女初长成》张一山为爱痴狂 甘心当保姆	http://ent.ifeng.com/a/20180
ifeng	_doc	wZ7N0GMBthV094w7jC3y	1	2018年06月05日 14:23:32	《新歌声》海选评委称导师周杰伦的歌曲没营养	http://ent.ifeng.com/a/20180
ifeng	_doc	w57N0GMBthV094w7jS30	1	2018年06月05日 12:48:56	《踢球吧少年强》名单出炉 名誉队长关晓彤现场送祝福	http://ent.ifeng.com/a/20180
ifeng	_doc	x57N0GMBthV094w7ki1z	1	2018年06月05日 19:38:24	34岁香港视帝被曝离婚，做这件事让老婆更加寒心	http://ent.ifeng.com/a/20180
ifeng	_doc	yp7N0GMBthV094w7li3P	1	2018年06月05日 18:41:28	杨子否认签署7.5亿阴阳合同：从未威胁过崔永元	http://ent.ifeng.com/a/20180
ifeng	_doc	Jp7N0GMBthV094w7_y7L	1	2018年05月07日 15:19:46	人气偶像现身跑男火爆苏州 竟然不敌网红实力抢镜	http://ent.ifeng.com/a/20180
ifeng	_doc	657N0GMBthV094w7wy1s	1	2018年06月05日 14:35:07	范冰冰和成龙旧照被扒 因为一个动作被网友骂惨了	http://ent.ifeng.com/a/20180
ifeng	_doc	4p7N0GMBthV094w7uC2R	1	2018年06月05日 08:45:13	《暹罗决：九神战甲》定档6.29 泰拳动画点燃今夏	http://ent.ifeng.com/a/20180
ifeng	_doc	5p7N0GMBthV094w7vi1Y	1	2018年06月05日 13:01:43	付辛博为寻爱犬自曝家庭地址和电话 网友：有勇气！	http://ent.ifeng.com/a/20180
ifeng	_doc	757N0GMBthV094w7yC3K	1	2018年06月05日 13:43:01	崔永元曝冯小刚洛杉矶有两套房产：当面一套背后一套	http://ent.ifeng.com/a/20180
ifeng	_doc	8p7N0GMBthV094w7zS2z	1	2018年06月05日 07:11:06	6岁的小七穿制服上学 紧牵贝嫂手抬头对视太温馨	http://ent.ifeng.com/a/20180

图 4.13　索引库中的数据

4.8.4　搜索模块

本节使用 Java High Level RESTful Client 构造一个检索请求。首先，使用 RestHighLevelClient 实例化对象的 SearchRequest()方法设置索引名称，然后在 sourceBuilder. query()方法中填

入具体查询方法的参数。这里使用 SearchSourceBuilder 类的实例来构建查询。在 from()和 size()方法中填写对索引库进行检索的游标开始位置和返回结果集的总数,最后使用 SearchResponse 类的实例来处理搜索结果,如代码段 4.62 所示。

代码段 4.62: 在 news 索引库中检索数据的实现

```
//import 语句,略
RestHighLevelClient client= new RestHighLevelClient(
        RestClient.builder(new HttpHost("localhost", 9200, "http")));
SearchRequest searchRequest= new SearchRequest("ifeng");
SearchSourceBuilder sourceBuilder= new SearchSourceBuilder();
sourceBuilder.query(QueryBuilders.wildcardQuery("title", "* 周杰伦 * "));
sourceBuilder.from(0);
sourceBuilder.size(5);
searchRequest.source(sourceBuilder);
SearchResponse searchResponse= client.search(searchRequest);
SearchHits hits= searchResponse.getHits();
SearchHit[] searchHits= hits.getHits();
for(SearchHit hit : searchHits) {
    System.out.println(hit.getSourceAsString());
}
client.close();
```

结合实际需要,这里用到了 wildcardQuery()方法。采用"通配符"+"关键词"+"通配符"的形式,可满足用户输入任意关键词均能检索信息的要求。图 4.14 是当搜索关键词为"周杰伦"时在控制台显示的部分结果。

"title":"《新歌声》海选评委称导师周杰伦的歌曲没营养","url"
"title":"昆凌与助手穿"情侣装" 周杰伦评论醋意十足","url":"
"title":"周杰伦方文山领衔歌坛八对黄金搭档 最后一对成绝响","
"title":"周杰伦中分头新造型 意外撞脸蔡依林"天菜"孔刘","url
"title":"王俊凯2018首支单曲合作方文山 透露想再和周杰伦合作"

图 4.14 搜索结果

4.8.5 推荐模块

新闻网站往往有相关新闻推荐功能。这里也可以通过 moreLikeThisQuery()实现,可通过它查询到与当前查询到的文本相似的文档返回给用户,如代码段 4.63 所示。

代码段 4.63: 使用 moreLikeThisQuery()方法实现推荐功能

```
//import 语句,略
String[] fields= {"title", "content"};
```

```
String[] texts={"音乐力作正式启航"};
MoreLikeThisQueryBuilder.Item[] items=null;
QueryBuilder qb=moreLikeThisQuery(fields, texts, items)
    .minTermFreq(1)
    .maxQueryTerms(12);
SearchResponse response=ESClientHelper.client.prepareSearch("ifeng")
    .setTypes("_doc")
    .setSearchType(SearchType.DFS_QUERY_THEN_FETCH)
    .setQuery(qb)
    .setFrom(0)
    .setSize(5)
    .setExplain(true)
    .get();
for(SearchHit hit : response.getHits().getHits()) {
    System.out.println(hit.getSourceAsString());
}
```

与查询相类似，在 moreLikeThisQuery(fields, texts, items)中填入的参数分别表示要检索的字段、文本和词项。图 4.15 列出的结果是基于一篇关于流行音乐出版发行的新闻推荐的部分结果。

```
"title":"曾轶可全新单曲《I Need Love》携MV正式发布","
"title":"罗伯特·德尼罗再出山 将出演中国投资电影","url":
"title":"《酒干倘卖无》开演倒计时 你不得不看的五大理由",'
"title":"浙交原创交响乐《社戏》首演 融汇众多浙江戏曲元素"
"title":"方大同与音乐品牌东亚星光联手 拓展音乐演出新纪元"
```

图 4.15 推荐结果

4.8.6 聚合模块

在 ifeng 索引库中，不同发布时间的新闻数量不同。这里可以通过聚合的方法对不同发布时间的新闻进行统计。代码段 4.64 实现了对其中的新闻数据按月进行数量统计的功能。聚合结果如图 4.16 所示，该结果给出了每个月内发布的新闻数量。在以后的工作中，这些数据可以用来绘制每月发布新闻数量的统计图表。

代码段 4.64：使用 date_histogram 聚合执行每月新闻数量统计

```
//import 语句，略
//主类定义，略
private static final Logger logger= (Logger) LogManager.getLogger(ClientTest.class);
```

```
//主方法 main()定义,略
AggregationBuilder aggregation= AggregationBuilders
            .dateHistogram("agg")
            .field("publishTime")
            .dateHistogramInterval(DateHistogramInterval.MONTH);           //步长为 1个月
SearchResponse sr= ESClientHelper.client.prepareSearch("ifeng")
      .setTypes("_doc")
      .addAggregation(aggregation)
      .get();
Histogram agg= sr.getAggregations().get("agg");
for(Histogram.Bucket entry : agg.getBuckets()) {
      DateTime key= (DateTime) entry.getKey();
      String keyAsString= entry.getKeyAsString();
      long docCount= entry.getDocCount();
      logger.info("key [{}], date [{}], doc_count [{}]", keyAsString,
      key.getYear(), docCount);
}
```

```
Run  ClientTest
key [2017年10月01日 00:00:00], date [2017], doc_count [18]
key [2017年11月01日 00:00:00], date [2017], doc_count [34]
key [2017年12月01日 00:00:00], date [2017], doc_count [41]
key [2018年01月01日 00:00:00], date [2018], doc_count [58]
key [2018年02月01日 00:00:00], date [2018], doc_count [40]
key [2018年03月01日 00:00:00], date [2018], doc_count [59]
key [2018年04月01日 00:00:00], date [2018], doc_count [72]
key [2018年05月01日 00:00:00], date [2018], doc_count [174]
key [2018年06月01日 00:00:00], date [2018], doc_count [263]

Process finished with exit code 0
```

图 4.16 聚合结果

4.9 扩展知识与阅读

众所周知,中文词法分析是中文信息处理的基础与关键。在 Java 客户端设计全文检索程序时,往往需要指定词法分析器。限于篇幅,本章未对 analyzer 部分进行讲述。文献(高凯,2010)对低版本 Lucene 的 analyzer 部分进行了说明;文献(张华平,2014)对汉语词法分析(包括网络语言分析、新特征语言抽取、自动分类、自动聚类、自动摘要、关键词抽取)等进行了详细的分析,并给出了算法实现。文献(罗刚,2014)也对相关技术实现进行了说明。而有关 Maven 的背景资料,可以参阅文献(许晓斌,2011)。

4.10 本章小结

本章从简单的信息检索、统计等方面入手，给出使用Java和Python基于Elasticsearch提供的功能实现信息处理操作的具体方法。具体来说，一般首先要做的工作是将Elasticsearch节点实例化，然后构建索引文件，基于用户给定的查询项获取文档集合（当然也可以完成对文档的增、删、改等工作），并根据实际需求完成聚合等数据分析工作；在完成进一步处理（如分页、高亮、过滤等）后，将处理结果返回到前端页面。Java、Python等客户端可以基于Elasticsearch提供的API实现对应的信息处理等操作。首先需要引入相关的依赖包，在Java中可用Maven添加依赖，在Python中则使用pip包管理工具进行安装；第二是实例化Elasticsearch客户端；第三是设计查询，查询形式有多种，Java使用的是QueryBuilders以及AggregationBuilders来实现查询和聚合操作，Python查询类似于第3章中使用JSON构建的查询语句。Java High Level RESTful Client和Elasticsearch DSL是官方提供针对Java和Python的更为简单易用的客户端实现方式。无论是哪种实现方式，本质上都是通过构建JSON查询语句调用Elasticsearch提供的接口，从而实现对数据的操作。另外，如果需要，还可以设定分页查询。分页查询有两种方式：一种是指定from/size；另一种是使用scroll/size，可以用scrollID继续往下查询，但是这种分页方式必须在设定的时间范围内完成。



Elasticsearch 配置与集群管理

> Elasticsearch comes with reasonable defaults for most settings. Before you set out to tweak and tune the configuration, make sure you understand what are you trying to accomplish and the consequences. The primary way of configuring a node is via the elasticsearch.yml file. This template lists the most important settings you may want to configure for a production cluster.
>
> elasticsearch.yml

基于 Elasticsearch 可以完成很多和信息存储、检索等相关的任务。本章对 Elasticsearch 的配置、集群管理等进行说明，并对提高索引和查询效率的策略进行了简述。通过对本章的学习，能达到更好地配置和使用 Elasticsearch 的目的。

5.1 Elasticsearch 的部分基本配置

Elasticsearch 的大多数配置信息在 Elasticsearch 安装主目录下的 config/elasticsearch.yml 文件中，所有配置都可使用环境变量。其他的配置信息在同一目录下的日志配置文件 log4j2.properties 中，按普通 log4j2 配置项来设置即可。

elasticsearch.yml 负责设置服务器的默认状态，Elasticsearch 的大多数配置在该配置文件中完成。本节给出针对 elasticsearch.yml 的部分配置设置信息，具体如下：

（1）集群名称 cluster.name，例如“cluster.name：elasticsearch”。设置好以后，Elasticsearch 会自动发现在同一网段下的节点。如果在同一网段下有多个集群，可用这个属性来区分不同的集群。

（2）节点名称 node.name。Elasticsearch 启动时会自动创建节点名称，也可以在 node.name 中配置，例如“node.name："Master"”。指定节点名称有助于利用 API 访问具体的节点。虽然集群启动时会给每个节点初始化一个默认的名称，但仍然建议在这里手动设置节

点名称。

（3）节点是否为主节点（master）。每个节点都可被配置成主节点，默认值为 true，如"node. master：true"。该设置的目的是指定该节点是否有资格被选举为主节点，默认集群中的第一个节点为主节点，如果这个节点宕机，就会重新选举主节点。

（4）设置节点是否存储数据。默认值为 true，即设置 node. data 的值为"node. data：true"。如果希望节点只是一个主节点，但是不存储数据，则应当设置为代码段 5.1 所示的属性。

代码段 5.1：设置节点是主节点但不存储数据

```
node.master: true
node.data: false
```

如果要使节点只存储数据，但不是主节点，则应当设置为代码段 5.2 所示的属性。

代码段 5.2：设置节点不作为主节点，但存储数据

```
node.master: false
node.data: true
```

如果要使节点既不是主节点，也不存储数据，则应该设置为代码段 5.3 所示的属性。

代码段 5.3：设置节点既不是主节点也不存储数据

```
node.master: false
node.data: false
```

对部分相关配置的说明如下：

（1）node. attr. rack 用于设置机架编号，如 Rack1。

（2）可在 node. max_local_storage_nodes 中设置一台机器能运行的最大节点数目。

（3）path. conf：/path/to/conf 用于设置配置文件的存储路径，默认是 Elasticsearch 根目录下的 config 文件夹。

（4）path. data：/path/to/data 用于设置分配给当前节点的索引数据所在位置，默认是 Elasticsearch 根目录下的 data 文件夹，可以选择包含一个以上的位置，用逗号隔开，这样使得数据在文件级别可跨越位置，在创建时就有更多的自由路径可供选择。

（5）path. logs：/path/to/logs 用于设置日志文件所在位置，默认是 Elasticsearch 根目录下的 logs 文件夹。

（6）设置绑定的 IP 地址，可以是 IPv4 地址或 IPv6 地址。默认情况下 Elasticsearch 使用 0. 0. 0. 0 地址，并为 HTTP 传输开启 9200～9300 端口，为节点到节点的通信开启 9300～

9400 端口。也可自行设置 IP 地址，可在配置文件的 network. bind_host 和 network. publish_host 中进行设置。

(7) transport. tcp. port 用于设置节点与其他节点交互的 TCP 端口，默认为 9300。

(8) transport. tcp. compress 用于设置是否压缩 TCP 传输时的数据，默认为 false。

(9) http. port 用于设置为 HTTP 传输监听定制的端口，默认为 9200。

(10) http. enabled 用于设置是否使用 HTTP 对外提供服务，默认为 true。

(11) http. max_content_length 用于设置内容的最大长度，默认为 100MB。

(12) discovery. zen. minimum_master_nodes 用于设置集群中的节点可以知道几个有主节点资格的节点，默认为 1。对于较大的集群来说，可以将该值设置为“具有主节点资格的节点数/2+1”。

(13) discovery. zen. ping_timeout 用于设置集群中自动发现其他节点时 ping 连接超时时间，默认为 3s。对于比较差的网络环境，可以提高该值来防止自动发现时出错。

(14) gateway. recover_after_nodes 用于设置集群中有几个节点启动时进行数据恢复，默认为 1。

(15) gateway. recover_after_time 用于设置初始化数据恢复进程的超时时间，默认为 5min。

(16) gateway. expected_nodes 用于设置这个集群中节点的数量，默认为 2。一旦有设定数量的节点启动，就会立即进行数据恢复。

(17) cluster. routing. allocation. node_initial_primaries_recoveries 用于初始化数据恢复时并发恢复线程的个数，默认为 4。

(18) cluster. routing. allocation. node_concurrent_recoveries 用于设置添加删除节点或负载均衡时并发恢复线程的个数，默认为 4。

(19) indices. recovery. max_bytes_per_sec 用于设置数据恢复时限制的带宽，如 100MB/s，默认为 0(即无限制)。

(20) discovery. zen. ping. unicast. hosts: ["host1","host2: port","host3[portX-portY]"]用于设置集群中主节点的初始列表，可通过这些节点自动发现新加入集群的节点。

> Tips 在 Elasticsearch 5.0 及后续版本中，不允许在节点配置文件中写入形如 number_of_shards、number_of_replicas 这样的索引级配置。应使用 RESTful index API 来更新所有节点的配置，关于这部分的执行方法参见 2.3 节的相关内容。

5.2 索引文件和查询优化

Elasticsearch 的索引文件是基于倒排索引机制完成的。从索引优化的角度出发，在建立索引文件时，要考虑到影响索引速度的以下因素：

- 分片数量。
- 节点数量。
- 索引操作（如合并、优化、索引写操作等）。
- 磁盘 I/O 次数及速度。

从提高效率的角度出发，可以从如下几点来考虑提高索引工作效率：

- 客户端减少频繁的连接并提高效率（如在可能的前提下使用 TCP 长连接、采用多线程机制、建立连接池等）。
- 尽量减小索引文件大小（索引前预处理、过滤等）。
- 合理规划映像。
- 合理使用分词。

从查询优化的角度来说，可以合理规划索引和分片来提高查询效率。

除了上述策略外，路由选择也是经常要用到的。由于 Elasticsearch 的信息往往分布在不同的分片中，因此在搜索时，大多数情况下需要遍历所有分片以便能检索到相关信息。其实，在某些情况下，如果能指定特定的分片（即显式地指定路由），有时是能够提高检索效率的。可以在存储数据时为每个文档自定义一个 routing 值来代替其 id 号，在查询、删除、更新数据时只需给出相同的 routing 值即可。在 Elasticsearch 中，对同样的 id 号会得到同样的哈希值，因此特定用户的所有文档会被放置在一个分片上。在检索时，利用哈希值就只需搜索一个分片而非遍历所有分片。代码段 5.4 给出了在索引文件 information 中加入新数据时指定 routing 的方法。

代码段 5.4：routing 参数的使用

```
curl -H 'Content-Type: application/json' -XPUT localhost:9200/information/
_doc/6?routing=user1&refresh=true -d '{
"title": "A new book"
}'
```

使用 routing 参数查询该条数据的代码如下：

```
curl -H 'Content-Type: application/json' -XGET 'localhost:9200/information/
_doc/6?routing=user1'
```

5.3 监控集群状态

可以通过 HTTP 接口监控集群状态，例如 http://localhost:9200/_cluster/health?pretty，可以观察到当前集群的状态。图 5.1 和图 5.2 是两个不同的 Elasticsearch 集群的状态。对一个 Elasticsearch 集群来说，其 status 的取值可以有如下几种：

- Green：当 Elasticsearch 能够根据配置分配所有分片和副本时的状态值，如图 5.1 所示。
- Yellow：主分片已经分配完毕，已经做好可以处理请求的准备，但是某些副本尚未完成分配。例如，当只有一个节点，却同时有多个副本时，因为尚无其他节点放置这些副本，因此其状态值可能就是 yellow，如图 5.2 所示。

```
{
  "cluster_name" : "elasticsearch",
  "status" : "green",
  "timed_out" : false,
  "number_of_nodes" : 2,
  "number_of_data_nodes" : 2,
  "active_primary_shards" : 30,
  "active_shards" : 60,
  "relocating_shards" : 0,
  "initializing_shards" : 0,
  "unassigned_shards" : 0,
  "delayed_unassigned_shards" : 0,
  "number_of_pending_tasks" : 0,
  "number_of_in_flight_fetch" : 0,
  "task_max_waiting_in_queue_millis" : 0,
  "active_shards_percent_as_number" : 100.0
}
```

图 5.1 状态为 green 的集群

```
{
  "cluster_name" : "elasticsearch",
  "status" : "yellow",
  "timed_out" : false,
  "number_of_nodes" : 1,
  "number_of_data_nodes" : 1,
  "active_primary_shards" : 30,
  "active_shards" : 30,
  "relocating_shards" : 0,
  "initializing_shards" : 0,
  "unassigned_shards" : 30,
  "delayed_unassigned_shards" : 0,
  "number_of_pending_tasks" : 0,
  "number_of_in_flight_fetch" : 0,
  "task_max_waiting_in_queue_millis" : 0,
  "active_shards_percent_as_number" : 50.0
}
```

图 5.2 状态为 yellow 的集群

- Red：集群目前尚未准备就绪，可能至少一个主分片没有准备好。

在 Elasticsearch 集群 IP 地址后加上/_cluster/health?pretty 参数，再加上 &level=indices 参数（例如，http://localhost:9200/_cluster/health?pretty&level=indices，如图 5.3 所示）或者 &level = shards 参数（例如，http://localhost：9200/_ cluster/health?pretty&level=shards，如图 5.4 所示），可以返回更详细的集群状态信息。图 5.4 不仅有索引文件的更详细的状态信息，还有其分片的状态信息。

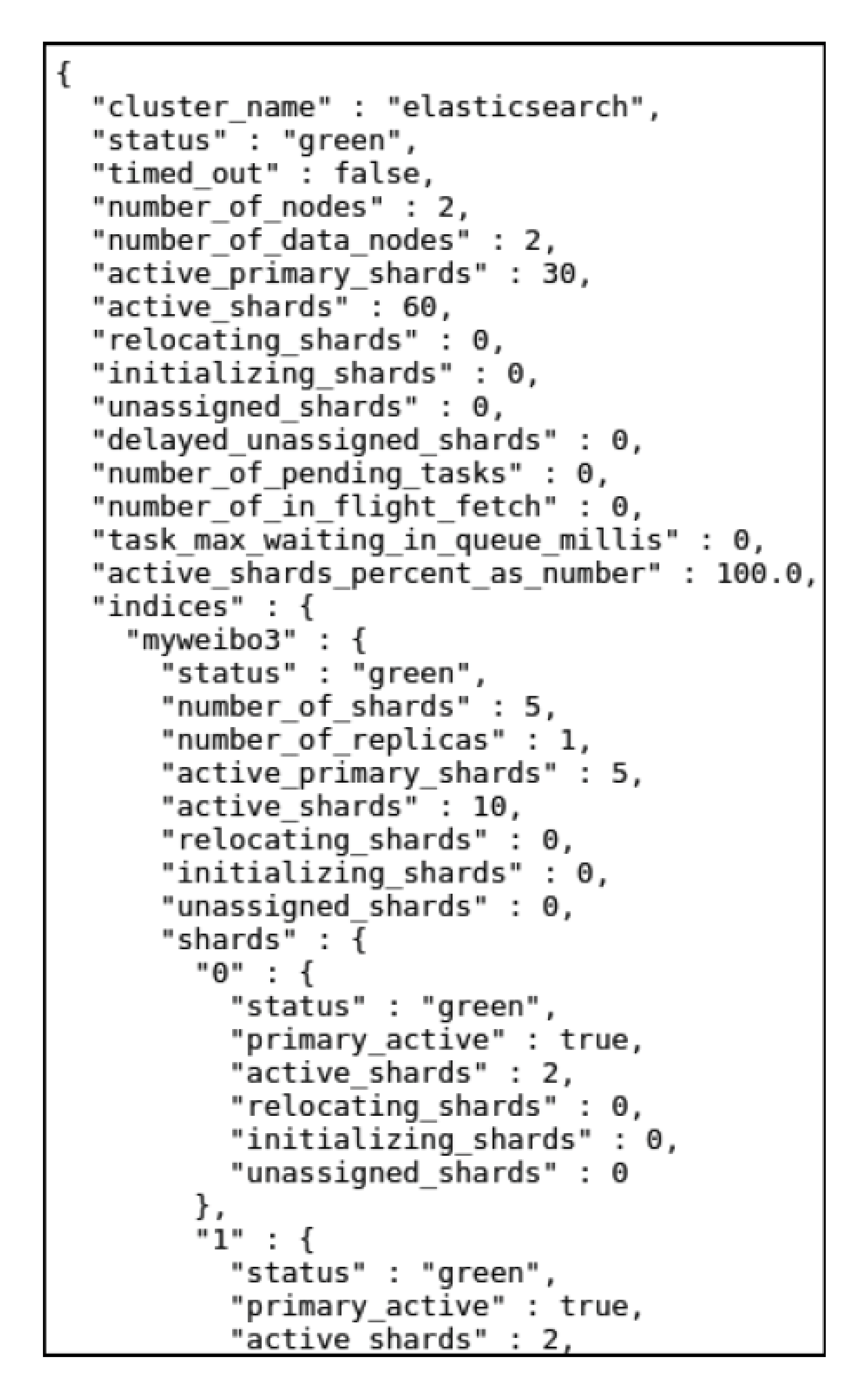
类似地，也可以监控节点状态，以了解集群在工作过程中发生了什么。和监控集群状态类似，只需在 URL 后给出相应节点的信息，例如如果监控 master 节点（master 是已在 elasticsearch.yml 中命名的节点名称），只需在 URL 的/_nodes 参数后添加节点名称及要查询的统计信息即可，如 http://localhost:9200/_nodes/master/stats/os,jvm?pretty，如图 5.5 所示。可以直接指明的部分可用标记信息如下：

- indices：获得分片大小、文件数量、索引的创建和删除时间、搜索执行时间、字段缓存

大小、数据合并与清缓存等信息。

```
{
  "cluster_name" : "elasticsearch",
  "status" : "green",
  "timed_out" : false,
  "number_of_nodes" : 2,
  "number_of_data_nodes" : 2,
  "active_primary_shards" : 30,
  "active_shards" : 60,
  "relocating_shards" : 0,
  "initializing_shards" : 0,
  "unassigned_shards" : 0,
  "delayed_unassigned_shards" : 0,
  "number_of_pending_tasks" : 0,
  "number_of_in_flight_fetch" : 0,
  "task_max_waiting_in_queue_millis" : 0,
  "active_shards_percent_as_number" : 100.0,
  "indices" : {
    "myweibo3" : {
      "status" : "green",
      "number_of_shards" : 5,
      "number_of_replicas" : 1,
      "active_primary_shards" : 5,
      "active_shards" : 10,
      "relocating_shards" : 0,
      "initializing_shards" : 0,
      "unassigned_shards" : 0
    },
    "weibo" : {
      "status" : "green",
      "number_of_shards" : 5,
      "number_of_replicas" : 1,
      "active_primary_shards" : 5,
      "active_shards" : 10,
      "relocating_shards" : 0,
      "initializing_shards" : 0,
      "unassigned_shards" : 0
    },
```

图 5.3 增加参数 level=indices 返回的集群状态信息



```
{
  "cluster_name" : "elasticsearch",
  "status" : "green",
  "timed_out" : false,
  "number_of_nodes" : 2,
  "number_of_data_nodes" : 2,
  "active_primary_shards" : 30,
  "active_shards" : 60,
  "relocating_shards" : 0,
  "initializing_shards" : 0,
  "unassigned_shards" : 0,
  "delayed_unassigned_shards" : 0,
  "number_of_pending_tasks" : 0,
  "number_of_in_flight_fetch" : 0,
  "task_max_waiting_in_queue_millis" : 0,
  "active_shards_percent_as_number" : 100.0,
  "indices" : {
    "myweibo3" : {
      "status" : "green",
      "number_of_shards" : 5,
      "number_of_replicas" : 1,
      "active_primary_shards" : 5,
      "active_shards" : 10,
      "relocating_shards" : 0,
      "initializing_shards" : 0,
      "unassigned_shards" : 0,
      "shards" : {
        "0" : {
          "status" : "green",
          "primary_active" : true,
          "active_shards" : 2,
          "relocating_shards" : 0,
          "initializing_shards" : 0,
          "unassigned_shards" : 0
        },
        "1" : {
          "status" : "green",
          "primary_active" : true,
          "active_shards" : 2,
```

图 5.4 增加参数 level=shards 返回的集群状态信息

- fs：获得文件系统信息、数据文件路径、可用磁盘空间信息、读/写状态等。
- http：获得 HTTP 连接信息。
- jvm：获得 Java 虚拟机的内存、垃圾回收、缓冲池、加载/未加载类的数量等信息。
- os：获得服务器的负载、内存使用情况、虚拟内存使用情况等统计信息。
- process：获得进程的内存开销、CPU 使用情况、打开文件描述符等统计信息。
- thread_pool：获得分配给不同操作的线程状态信息。
- transport：获得关于传输模块发送和接收数据的信息。
- breaker：获得 field data 断路器的统计信息。
- discovery：获得第三方集群的统计信息。
- ingest：获得 ingest 预处理的统计信息。

```
{
  "_nodes" : {
    "total" : 1,
    "successful" : 1,
    "failed" : 0
  },
  "cluster_name" : "cyElasticsearch",
  "nodes" : {
    "SDSFWxIaTzKgWeOv_GmncQ" : {
      "timestamp" : 1528637931465,
      "name" : "Master",
      "transport_address" : "127.0.0.1:9300",
      "host" : "127.0.0.1",
      "ip" : "127.0.0.1:9300",
      "roles" : [
        "master",
        "data",
        "ingest"
      ],
      "attributes" : {
        "rack" : "Rack1"
      },
      "os" : {
        "timestamp" : 1528637931466,
        "cpu" : {
          "percent" : 28,
          "load_average" : {
            "1m" : 0.45,
            "5m" : 0.33,
            "15m" : 0.28
          }
        },
        "mem" : {
          "total_in_bytes" : 16731815936,
          "free_in_bytes" : 11800961024,
          "used_in_bytes" : 4930854912,
          "free_percent" : 71,
          "used_percent" : 29
        },
```


图 5.5　节点状态信息

5.4　控制索引文件分片与副本分配

集群中的索引文件可能由多个分片构成，且每个分片可能拥有多个副本。通过将一个单独的索引文件分成多个分片，可以有效处理由于索引文件太大而不能在一台机器上运行的问题。由于每个分片可以有多个副本，通过将副本分配到多个服务器上可以处理更高的查询负载。为了进行分片和副本分配操作，Elasticsearch 需要确定将这些分片和副本放在集群的什么地方，即需要确定每个分片和副本分配到哪一个服务器/节点上（Rafal，2015）。可以对一个索引指定每个节点上的最大分片。例如，如果希望 it-home 索引文件在每个节点上有两个分片，可以运行代码段 5.5。

代码段 5.5：指定每个节点上的最大分片数

```
curl -H 'Content-Type: application/json' -XPUT 'localhost:9200/it-home/
_settings' -d '{
"index.routing.allocation.total_shards_per_node" : 2
}'
```

这个属性可以放置在 elasticsearch.yml 文件中，或使用上面的命令在活动索引上更新(Rafal，2015)，也可以手动移动分片和副本。可以使用 Elasticsearch 提供的 RESTful 代码 _cluster/reroute 进行控制。

假设有 3 个节点 node1、node2 和 node3，Elasticsearch 在 node1 上分配了 it-home 索引文件的两个分片，假设希望将第二个分片移动到 node2 上，并在 node3 上创建副本，可以采用如代码段 5.6 所示的方式。

代码段 5.6：移动分片和创建副本

```
curl -H 'Content-Type: application/json' -XPOST 'localhost:9200/_cluster/reroute' -d '{
  "commands": [
    {
      "move": {                          //移动分片
        "index": "it-home",              //待移动的索引文件
        "shard": 1,                      //指定希望移动的分片个数
        "from_node": "node1",            //指定源节点
        "to_node": "node2"               //指定目的节点
      }
    },
    {
      "allocate_replica": {
        "index": "it-home",              //待创建副本的索引文件
        "shard": 1,                      //指定希望创建副本的分片个数
        "node": "node3"                  //创建副本的节点
      }
    }
  ]
}'
```

可以通过运行 cancel 命令来指定希望取消分配的索引、节点以及分片，如代码段 5.7 所示。

代码段 5.7：取消分片分配

```
curl -H 'Content-Type: application/json' -XPOST 'localhost:9200/_cluster/reroute' -d '{
  "commands": [
    {
```

```
            "cancel": {                     //取消操作
              "index": "it-home",           //指定索引文件
              "shard": 1,                   //指定要取消的分片
              "node": "node1"               //指定取消分配的节点
            }
          }
        ]
      }'
```

另外,还可以将一个未分配的分片分配到一个指定的节点上。假定 it-home 索引文件中有一个编号为 1 的分片尚未分配,现希望将其分配到 node2 上,可使用代码段 5.8 的方式来实现。

```
代码段 5.8: 分配分片
curl -H 'Content-Type: application/json' -XPOST 'localhost:9200/_cluster/reroute' -d ' {
  "commands": [
    {
      "allocate": {                     //分配分片
        "index": "it-home",             //指定索引文件
        "shard": 1,                     //指定要分配的分片编号
        "node": "node2"                 //指定节点
      }
    }
  ]
}'
```

5.5 集群管理

下面以某个实际环境中的集群管理为例,介绍内存配置等方面的内容。集群硬件采用 4 台戴尔 PowerEdge R720 机架服务器,操作系统为 Ubuntu 16.04,Elasticsearch 版本为 6.2。

首先,在每台机器上安装 Elasticsearch 并手动安装 X-Pack 插件(详见 8.2 节)、中文分词器等必要的插件(可能需要在 elasticsearch.yml 中完成配置)。

以后台启动的方式分别运行每台机器上装有 X-Pack 插件的 Elasticsearch,并运行 Kibana(详细介绍参见第 7 章)。进入 Kibana 中的 Monitoring 界面(即以前版本中的

Marvel)，单击 Elasticsearch 面板中的 Overview 链接，进入 Elasticsearch 的监控界面。在上方统计数据栏中显示默认内存大小是 4GB，这通常无法满足实际需要。此时需要修改 config 目录下的 jvm.options 中默认的参数-Xms2g 和-Xmx2g 来设置内存大小。修改完成后重启 Elasticsearch，再次进入 Monitoring 界面，从 Memory 的值可以看出修改已经生效，集群占用的总内存量上限为 32GB(修改前为 4GB)，如图 5.6 所示。

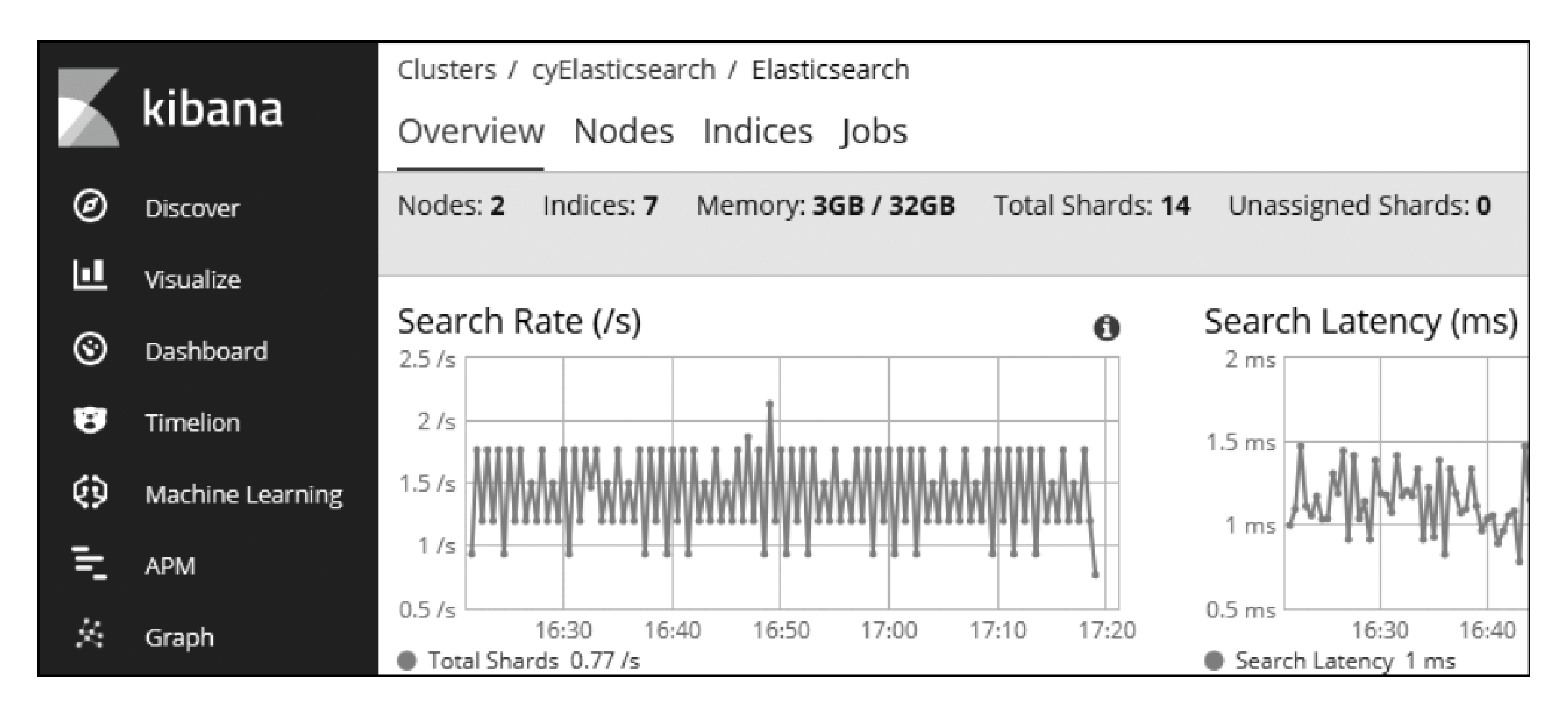


图 5.6 修改后的 Elasticsearch 内存大小

> Tips 在 Linux/Ubuntu 系统中，配置 Elasticsearch 默认的内存大小时，既可以使用上述方法，也可以在 bin 目录下的 Elasticsearch 程序（可用 gedit 等文本编辑工具打开）中取消 ES_JAVA_OPTS="-Xms8g -Xmx8g"的注释，并根据实际需要修改其数值，然后将 config 目录下 jvm.options 文件中的-Xms2g 和-Xmx2g 参数注释掉，并重启 Elasticsearch。

5.6 扩展知识与阅读

文献(Rafal，2015)介绍了路由选择方法、索引别名及其用途，给出了监控集群状态与健康状况的 API 的使用方法，对常用的集群诊断工具进行了介绍，对一些问题（如深翻复杂度优化、控制集群再平衡、验证查询等）的处理也给出了建议。

5.7 本章小结

本章首先介绍了 Elasticsearch 的基本配置。虽然默认的配置文件已经能够应付大多数情况,但了解有关 YML 配置文件的详细内容还是很有必要的。除此之外,本章介绍了提高索引和查询效率的策略(如介绍了 routing 参数的使用),这在一些情况下是有用的。通过监控集群状态,可以使管理员了解集群的整体运行情况,及时发现可能存在的问题。而手动控制索引分片与副本分配,能更有效地调节集群状态,在很多情况下是必要的。



基于 Logstash 的日志处理

> Logstash is an open source data collection engine with real-time pipelining capabilities. Logstash can dynamically unify data from disparate sources and normalize the data into destinations of your choice. Any type of event can be enriched and transformed with a broad array of input, filter, and output plugins, with many native codecs further simplifying the ingestion process. Logstash accelerates your insights by harnessing a greater volume and variety of data.
>
> https://www.elastic.co

随着大数据、大型综合网站以及 Web 2.0 技术的普及，越来越多的软件开发者需要处理海量的日志信息。网络搜索日志分析旨在帮助网络管理员或搜索引擎开发者提升网络管理质量，提高搜索性能。网络搜索日志记录的主要数据包括用户 id、查询关键词、网页点击行为、行为发生时间和用户 IP 地址等信息。通过对日志的分析，可以挖掘用户的搜索模式，理解用户的搜索意图，进而在网站建设、信息推荐、个性化服务等方面向用户提供更好的服务。不论是搜索系统还是广告或推荐系统，现在越来越多地使用甚至依赖日志信息来帮助提升系统性能。日志往往分散在各种服务/设备上，不同的服务/设备可能会有不同种类的日志。日志数据集的共享机制也是工业界和学术界共同努力的方向。一般来说，日志数量巨大，共享并非易事。人们期望不必通读日志或者不必理解日志中的数据格式，就能挖掘出隐藏在海量日志背后的知识。但对于通常的日志处理技术而言，目前的日志信息缺乏统一的格式，每个应用程序或设备都可以有自己的日志记录格式。日志数据格式众多，表达方式不同(如对时间戳的表示形式各异)，导致对日志的搜索与挖掘很困难。对这些非中心化的海量日志信息而言，高效地处理、挖掘其中的信息变得越来越重要。通常采用的解决办法是在中央系统上由专门的日志服务器收集日志，这样可帮助用户通过 SSH 协议在多台服务器之间浏览、查找信息。但传统的日志系统接入周期较长，有时不是任何字段都可以搜索的，处理速度也比较慢。另外，统计数据一般是预先聚合好的，无法按需实时计算，如果新增加

一个分析维度，往往要进行二次开发。当日志信息量越来越大时，从快速增长的海量日志数据流中提取所需信息，并将其与其他事件进行关联，将变得越来越困难。

Elastic Stack 中的 Logstash 是一个能有效进行日志处理的工具，可以对日志进行收集、分析。Logstash 本身并不产生日志，它只是一个内置了分析和转换工具的日志管理工具，是一个接收、处理、转发日志的管道。不同于分离的代理端或主机端，Logstash 可配置单一的代理端并与其他开源软件结合，以实现不同的功能。

Logstash 处理事件一般有 3 个阶段：输入、过滤和输出。输入插件 input 从数据源获取数据，过滤插件 filter 根据用户配置的数据格式修改数据，输出插件 output 将其转换并输出到其他位置。输入插件 input 和输出插件 output 均支持 Codecs，而 Codecs 使其能够编码或解码数据。在输入和输出阶段，可以对进入的或退出 Logstash 的数据进行编码或解码，在过滤阶段对日志进行分析、过滤。可见，Logstash 不但可以接收多种格式的日志输入，还可以解析日志，并将多种格式的日志输出到不同的目的地。例如，Logstash 可以将日志收集在一起交给 Elasticsearch，由 Elasticsearch 进行自定义搜索，并借助 Kibana 进行分析结果的展示。

6.1 概述

Logstash 提供了几百种插件，可以方便地处理几乎任何数据源中的各种日志数据。日志数据可以转换为各种数据类型输出，也能方便地采用二次开发的插件完善个性化处理。Logstash 6.x 和 Elasticsearch 6.x 是兼容的，提供离线插件安装、性能监控等功能，并且支持 Filebeat（一种安装在服务器上的代理程序，用于解析和传输特定目录中日志文件）。Logstash 的结构如图 6.1 所示。输入部分可以接收 tcp、file、syslog、kafka、rabbitmq（一种可复用的消息队列）、redis、stdin、twitter 等格式的数据；过滤部分包括 grok（使用模式匹配的数据抽取）、date（从字段中解析时间戳等）、csv、geoip（通过来访者的 IP 地址定位其地理位置等信息）、kv 键-值对、ruby 等；输出部分包括日志存储（如存储在 Elasticsearch、MongoDB、S3 及文件中）、通知（如警报系统工具 Pagerduty、开源的网络监视工具 nagios、提供分布式系统监视以及网络监视功能的 zabbix、Email 等）、消息队列（如 tcp、kafka、redis、rabbitmq、syslog 等）。Codec 用于对数据流进行某种格式化处理，如 json、msgpack、plain 等，它可作为输入输出流中的一部分，帮助分割或格式化数据。

一般地，需要创建一个含有定义输入输出等内容的配置文件，可以在文件内写入数据来源和目的地、是否对日志数据进行某种格式转换等，并在启动 Logstash 时指定拟采用的配置文件名称。

代码段 6.1 给出 Logstash 的一种简易的配置文件，这里的信息设置为从键盘输入数

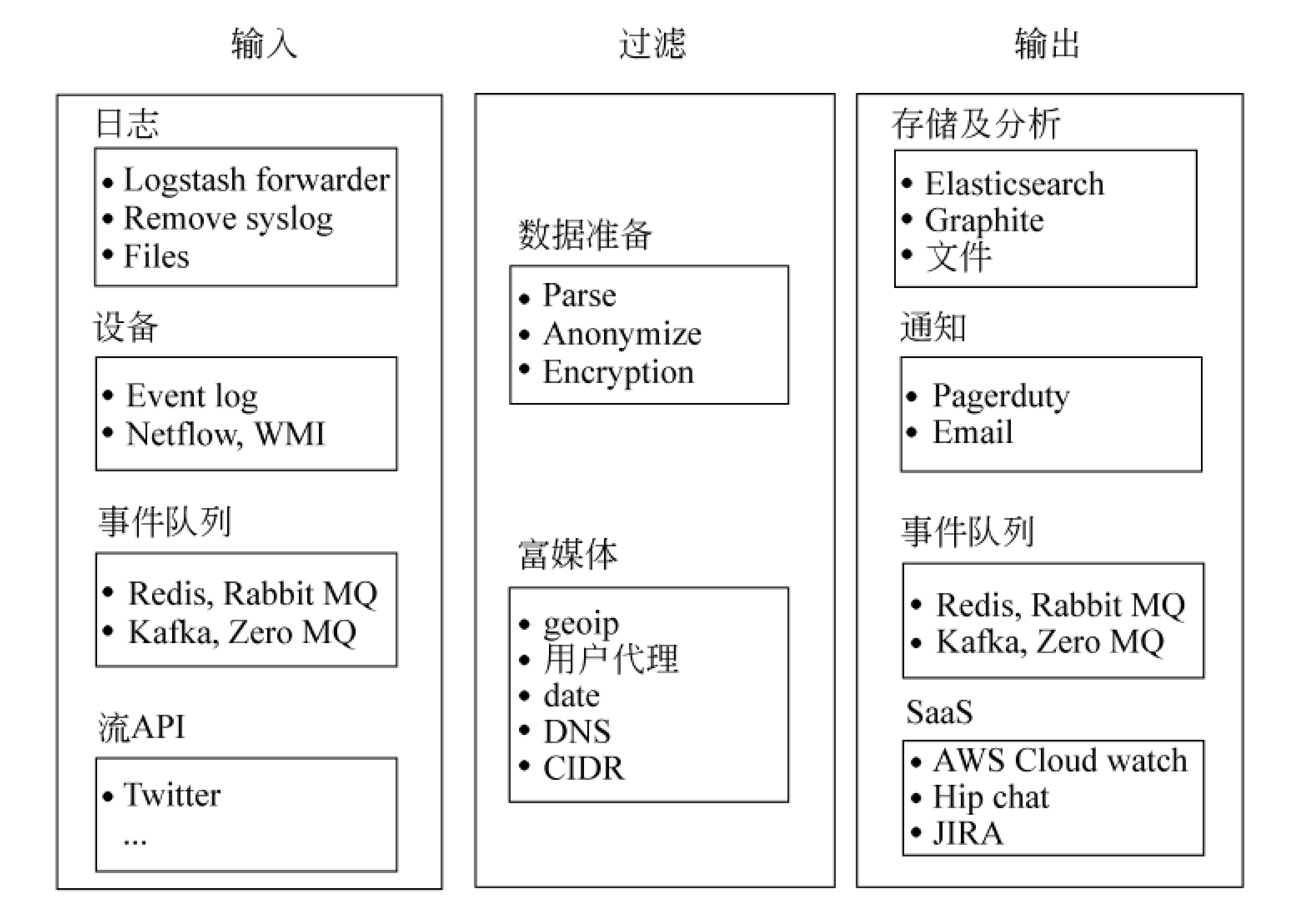


图 6.1　Logstash 的结构

据，输出的信息在屏幕上显示（Logstash 和 Kibana 相关配置文件中的代码采用＃注释的方式，＃后的文字是说明）。

代码段 6.1：在 Logstash 的配置文件中设置数据来源和目的地

```
input {              #数据来源
      stdin{}        #stdin是标准输入文件
}
output {             #目的地
      stdout{}       #stdout是标准输出文件
}
```

在 Linux 中可使用下面的命令启动 bin 目录下的配置文件（这里的启动参数-f 的作用是指定一个 Logstash 配置文件，此例为 conf. conf）：

```
./logstash -f ./conf.conf
```

也可使用如下命令从键盘接收数据，输出到屏幕上，效果与上面的命令一致。

```
./logstash -e 'input { stdin { } } output { stdout {} }'
```

Tips 如果在 Windows 系统中运行上述命令，需要保证配置文件为无 BOM 的 UTF-8 格式文件。Logstash 还可附加其他参数，如上述命令中的-e 参数，其作用是执行引号里的设置。也可通过-n 设置节点名称，通过-p 设置插件目录，通过-l 设置日志路径等。除此之外，还有很多参数，可使用 logstash -help 命令查看参数说明或浏览 Logstash 官方文档。

然后，系统会等待用户输入。用户可以通过键盘输入一些字符，输入完毕并按回车键之后，屏幕上会出现如图 6.2 所示的内容(其中的“hello world”字符串是用户输入的测试内容)。从返回的内容中可以看到，其输出结果为 JSON 格式的数据，其中包含了用来标记事件的发生时间的@timestamp、标记事件发生所在位置的 hostname(图 6.2 的测试环境中的 hostname 是 localhost. localdomain)、事件的版本以及用户输入的内容。如果能看到类似的结果，说明 Logstash 已经可以正常运行了。

```
shiyanshi@507:~/jiangyuehua/logstash-6.2.4/bin$ ./logstash -e 'input { stdin { } } output { stdout {} }'
Sending Logstash's logs to /home/shiyanshi/jiangyuehua/logstash-6.2.4/logs which is now configured via l
[2018-07-05T08:51:11,555][INFO ][logstash.modules.scaffold] Initializing module {:module_name=>"netflow"
on"}
[2018-07-05T08:51:11,597][INFO ][logstash.modules.scaffold] Initializing module {:module_name=>"fb_apach
ration"}
[2018-07-05T08:51:12,677][WARN ][logstash.config.source.multilocal] Ignoring the 'pipelines.yml' file be
[2018-07-05T08:51:13,866][INFO ][logstash.runner          ] Starting Logstash {"logstash.version"=>"6.2.
[2018-07-05T08:51:15,037][INFO ][logstash.agent           ] Successfully started Logstash API endpoint {
[2018-07-05T08:51:19,353][INFO ][logstash.pipeline        ] Starting pipeline {:pipeline_id=>"main", "pi
[2018-07-05T08:51:19,563][INFO ][logstash.pipeline        ] Pipeline started successfully {:pipeline_id=
The stdin plugin is now waiting for input:
[2018-07-05T08:51:19,726][INFO ][logstash.agent           ] Pipelines running {:count=>1, :pipelines=>["
hello world
{
          "host" => "507",
       "message" => "hello world",
    "@timestamp" => 2018-07-05T00:51:51.509Z,
      "@version" => "1"
}
```

图 6.2　Logstash 运行界面

Tips Logstash 用{}来定义区域，支持常用的数据类型。Logstash 的 DSL——例如区域、注释、数据类型(布尔值，字符串，数值，数组，哈希)、条件判断、字段引用等——是 Ruby 风格的。

6.2　input：处理输入的日志数据

可以把 Logstash 看成对日志信息进行处理的管道。由于日志数据的来源是多种多样的，因此要在 Logstash 中的 input 部分设置对不同来源的数据进行处理的方法。

Tips Logstash 可以多个数据源作为日志输入端，例如，可设置 beats、cloudwatch、couchdb_changes、drupal_dblog、elasticsearch、eventlog、exec、file、ganglia、gelf、gemfire、generator、github、graphite、heartbeat、heroku、http、http_poller、imap、irc、jdbc、jmx、kafka、kinesis、log4j、lumberjack、meetup、pipe、puppet_facter、rabbitmq、rackspace、redis、relp、rss、s3、salesforce、snmptrap、sqlite、sqs、stdin、stomp、syslog、tcp、twitter、udp、unix、varnishlog、websocket、wmi、xmpp、zenoss、zeromq 等多种来源的数据作为输入。通过使用过滤机制，编程人员可以修改、操纵这些数据和事件。限于篇幅，本章只对部分插件的可用参数、配置和命令进行简介。

6.2.1 处理基于 file 方式输入的日志信息

通过 file 方式，可以从指定的文件中读取数据并输入到 Logstash 中。基于这一特性，可以监控某些程序的日志文件(要求这些日志文件的格式是以行来组织的)。例如，可以将代码段 6.1 修改为代码段 6.2。

代码段 6.2: Logstash 配置文件格式，基于 file 方式读取指定文件

```
input {
  file {
    codec=>…                       #可选项,默认是 plain,可通过这个参数设置编码方式
    discover_interval=>…           #可选项,指定 Logstash 隔多久检查被监听的路径下是否有新
                                   #文件,默认为 15s
    exclude=>…                     #可选项,不想被监听的文件可在这里指定
    path=>…                        #必选项,指定需处理的文件路径
    sincedb_path=>…                #可选项,如果不想用默认的 $HOME/.sincedb* ,可在这里指
                                   #定其他位置的配置文件
    sincedb_write_interval=>…
                                   #可选项,指定 Logstash 隔多久写一次 sincedb 文件,默认为 15s
    start_position=>…              #可选项,取值范围为 ["beginning", "end"],指定 Logstash
                                   #从什么位置开始读取文件数据,默认为 end。如要导入原有数
                                   #据,将其改成 beginning,Logstash 就从头开始读取
    stat_interval=>…               #可选项,指定 Logstash 隔多久检查一次被监听文件状态是否
                                   #有更新,默认为 1s
    tags=>…                        #可选项
    type=>…                        #可选项
    # 其他选项,略
  }
}
```


在代码段 6.2 中，除 path 为必选项以外，其他参数都是可选项。

依照这种方式，这里给出一个配置文件实例，见图 6.3。在这里指定了待处理的日志文件的路径(其中的参数 start_position 设为 beginning，意为从该文件的头开始处理)。输出目的地包括 Elasticsearch。在图 6.4 中显示文件内容已经输出到 Elasticsearch 的索引文件中。

```
input {
    file {
        path => "/home/shiyanshi/jiangyuehua/logstash-6.2.4/NOTICE.TXT"
        start_position => "beginning"
        sincedb_path => "/home/shiyanshi/jiangyuehua/logstash-6.2.4/my.txt"
    }
}
output {
    stdout{}
    elasticsearch {              #通过HTTP的方式将数据传输到Elasticsearch中
        hosts => ["localhost"]   #指定数组形式的hosts列表
    }
}
```

图 6.3 conf.conf 配置文件


```
{
    "_index": "logstash-2018.07.02",
    "_type": "doc",
    "_id": "DOWmWWQBmC1EBcTh52DM",
    "_version": 1,
    "_score": 1,
    "_source": {
        "message": "The above copyright notice and this permission notice shall be included
        in all ",
        "path": "/home/shiyanshi/jiangyuehua/logstash-6.2.4/NOTICE.TXT",
        "host": "507",
        "@version": "1",
        "@timestamp": "2018-07-02T06:21:05.468Z"
    }
}
```


图 6.4 输出到 Elasticsearch 的索引文件中的信息

6.2.2 处理基于 generator 产生的日志信息

generator 可生成指定的或随机的日志信息。代码段 6.3 给出在 input 段中使用 generator 的方法，数据输出则使用 stdout 方法在屏幕上输出，实际效果如图 6.5 所示。图 6.5 的左侧是配置文件，请注意这里 count 和 lines 的用法；图 6.5 右侧是部分数据的实际执行结果。

代码段 6.3：Logstash 配置文件格式，基于 generator 的日志输入

```
input {
  generator {
    codec=>…                    #可选项，默认为 plain
    count=>…                    #可选项,可设置发送多少次,默认为 0(即不限制)
    lines=>…                    #可选项,可以数组格式顺序发送
    message=>…                  #可选项,可写成指定字符串,默认为"Hello world!"
    tags=>…                     #可选项,标记日志,可用于后续处理工作
    threads=>…                  #可选项,可以设置用来发送日志信息的线程数,默认为 1
    type=>…                     #可选项,默认为 string 类型
  }
}
```

```
input {
    generator {
        codec => "plain"
        count => 1
        lines => ["line-1","line2"]
        tags => ["tag1"]
        threads => 1
    }
}
output{
    stdout {}
}
~
~
~
~
~
~
~
```

```
[2018-07-05T09:32:01,394][INFO ][logstash.agent          ] Pipelines running {:co
{
       "message" => "line-1",
      "@version" => "1",
    "@timestamp" => 2018-07-05T01:32:01.325Z,
      "sequence" => 0,
          "host" => "507",
          "tags" => [
        [0] "tag1"
    ]
}
{
       "message" => "line2",
      "@version" => "1",
    "@timestamp" => 2018-07-05T01:32:01.441Z,
      "sequence" => 0,
          "host" => "507",
          "tags" => [
        [0] "tag1"
    ]
}
```

图 6.5　基于 generator 的日志输入及实际输出

6.2.3　基于 Filebeat 处理 log4j 的日志信息

在 Logstash 的配置文件的输入部分配置有关 beats 的部分，并在 Filebeat 中配置相关的输入输出信息，可使 Filebeat 读取 log4j 中产生的日志信息数据并传输至 Logstash 中。代码段 6.4 给出在 Logstash 中处理 beats 的传输参数的方法。

代码段 6.4：Logstash 配置文件格式，基于 log4j 的日志输入

```
input {
  beats {
    codec=>…                    #可选项,默认为 plain
    host=>…                     #可选项,默认为 0.0.0.0
    cipher_suites=>…            #可选项,选择加密方式
```

```
        client_inactivity_timeout=>…  #可选项,空闲客户端关闭时间
        cipher_suites=>…              #可选项,选择加密方式
        ssl  =>…                          #boolean类型的可选项,设置是否开启加密传输,默认为否
        port=>…                       #必选项,无默认值
        tags=>…                       #可选项,标记日志,可用于后续处理工作
        type=>…                       #可选项,默认为 string类型
        #其他选项,略
    }
}
```

图 6.6 给出了部分 log4j 日志文件数据。代码段 6.5 指定了 Logstash 接收 beats 数据的端口,并将接收到的数据输出到屏幕上。代码段 6.6 给出使用 Filebeat 读取 log4j 所产生的数据并传送至 Logstash 中的配置信息。

代码段 6.5: Logstash 配置

```
input {
  beats {
    port=> 5000                  #配置接收输入的端口为 5000
  }
}
output{
  stdout{}                       #直接输出到屏幕上
}
```

代码段 6.6: Filebeat 配置

```
filebeat.inputs:
- type: log
  enabled: true
  paths:
    - C:\Users\84902\Desktop\testLOG.txt        #读取的文件路径
output:
  logstash:
    hosts: ["192.168.1.111:5000"]               #输出的目标机器 IP地址和端口
```

```
2017-11-21 15:33:57,821  INFO [servlet.mvc.annotation.DefaultAnnotationHandlerMapping:315] Mapped URL path [/system/ext
2017-11-21 15:33:57,821  INFO [servlet.mvc.annotation.DefaultAnnotationHandlerMapping:315] Mapped URL path [/system/ext
2017-11-21 15:33:57,821  INFO [servlet.mvc.annotation.DefaultAnnotationHandlerMapping:315] Mapped URL path [/system/ext
2017-11-21 15:33:57,821  INFO [servlet.mvc.annotation.DefaultAnnotationHandlerMapping:315] Mapped URL path [/system/ext
2017-11-21 15:33:57,824  INFO [servlet.mvc.annotation.DefaultAnnotationHandlerMapping:315] Mapped URL path [/system/neg
2017-11-21 15:33:57,825  INFO [servlet.mvc.annotation.DefaultAnnotationHandlerMapping:315] Mapped URL path [/system/neg
2017-11-21 15:33:57,825  INFO [servlet.mvc.annotation.DefaultAnnotationHandlerMapping:315] Mapped URL path [/system/neg
```

图 6.6 log4j 的部分日志数据

Tips 在 Logstash 6.x 以上版本中，已弃用 log4j 插件。官方推荐的使用方式是通过 Filebeat 读取日志文件并传输到 Logstash 中，这种方式可设置 SSL 加密传输，更推荐使用在生产环境中。但是，Logtash 的 log4j 插件仍可通过 bin/logstash-plugin install logstash-input-log4j 命令安装和使用，详情请查阅官方文档。

运行程序后，将基于 log4j 产生的输出数据作为 Logstash 的输入信息。图 6.7 是经过 Logstash 处理后的输出信息。

```
{
        "offset" => 1152,
          "host" => {
        "name" => "DESKTOP-ICGDDPS"
    },
        "source" => "C:\\Users\\84902\\Desktop\\log.txt",
    "prospector" => {
        "type" => "log"
    },
          "beat" => {
        "hostname" => "DESKTOP-ICGDDPS",
         "version" => "6.3.0",
            "name" => "DESKTOP-ICGDDPS"
    },
          "tags" => [
        [0] "beats_input_codec_plain_applied"
    ],
       "message" => "2018-04-21 15:33:57,825  INFO [servlet.mvc.annotation.
andler 'negativeAction'",
      "@version" => "1",
    "@timestamp" => 2018-07-05T01:50:04.576Z,
         "input" => {
        "type" => "log"
    }
}
```

图 6.7　经过 Logstash 处理后的输出信息

6.2.4　处理基于 redis 的日志信息

Logstash 可从 redis 实例中获取日志。通过在 Logstash 的配置文件中使用 redis 部分，可以将 redis 中的日志信息输出到 Logstash 中。

Tips redis 是一个键-值对存储系统，支持存储的值包括 string、list、set、hash 等类型。为了提高存储效率，数据都是缓存在内存中的。

代码段 6.7 给出了在 Logstash 端的配置文件格式。需要说明如下几点：

- data_type：在配置文件中通过设定它来指定 Logstash 即将处理的 redis 数据源是列表形式的还是 channel 或 pattern_channel 形式的。其中，pattern_channel 可指定

一个或多个给定模式的频道，例如 it * 匹配所有以 it 开头的频道（it. news、it. blog、it. tweets 等）。

- key：由于 redis 是一个基于键-值对的存储系统，因此在进行数据处理时，要指定 list 或 channel 的名字（也就是键）。redis 相当于消息的发布方，而 Logstash 则是消息的订阅方。

代码段 6.7：Logstash 配置文件格式，基于 redis 输入数据流

```
input {
  redis {
    add_field=>…          #哈希值,可选项,默认为{}
    batch_count=>…        #数值,可选项,默认为 1
    codec=>…              #编解码器,可选项,默认为 json
    data_type=>…          #字符串,必选项,取值为"list"、"channel"、"pattern_channel"之一
    db=>…                 #数值,可选项,默认为 0
    host=>…               #可选项,默认为 127.0.0.1
    path=>…               #可选项
    key=>…                #字符串,必选项
    password=>…           #可选项
    port=>…               #可选项,默认为 6379
    ssl   =>…             #可选项,指定是否使用加密
    tags=>…               #可选项,标记日志,可用于后续处理工作
    threads=>…            #数值,可选项, 默认为 1
    timeout=>…            #数值,可选项, 默认为 5
    type=>…               #可选项,默认为 string类型
  }
}
```

下面给出基于 redis 的日志数据应用实例。首先安装并配置好 redis，然后启动 redis 服务器端，在 redis 配置文件中设置服务器端密码、端口号等（默认端口号为 6379）。启动服务端后，可使用客户端进行测试。当确保 redis 正常启动以后，可在 redis 端设置日志数据流。

1. 测试基于列表的日志数据流

首先测试基于列表（list）的日志数据流。图 6.8 给出 Logstash 的配置文件。其中，参数 batch_count 指定从 redis 中一次读取两条事件日志，参数 data_type 指定输入的数据来源于列表，参数 key 指定列表的名字为 msg，password 指定此 redis 服务器端密码是 123456，其他的采用默认值即可。

```
input {
    redis {
        codec => "plain"
        batch_count => 2
        data_type => "list"
        host => "127.0.0.1"
        key => "msg"
        password => "123456"
    }
}
output{
    stdout {}
}
```

图 6.8 Logstash 的配置

在启动 Logstash 前，可先测试 redis 的运行情况。

图 6.9左侧的第一条命令是启动 redis 客户端，然后通过 rpush 命令向名为 msg 的列表中插入 3 个值(分别是"hello world"、"hello Logstash"、"hello elasticsearch")，它们被依次插入 msg 中。



rpush 是 redis 的命令，其作用是将一个或多个值插入列表的表尾(最右边)。

然后，启动 Logstash，可以看到 redis 中的数据(即 msg 中的"hello world"、"hello Logstash"、"hello elasticsearch"等字符串)已经通过 Logstash 显示在屏幕上，如图 6.9 右侧所示。此时 msg 中的数据流被清空。之后，通过 rpush 命令输入新的字符串"redis"(此时 msg 中的数据流数量是 1 条)。由于此时 Logstash 已启动，因此这个输入的数据(即"redis"字符串)会经 Logstash 显示在屏幕上。此时，redis 端的 msg 中的数据流被清空(可使用 lrange msg 0 -1 命令查看 msg 中的值)，数据流已经通过 Logstash 这个管道流向了屏幕端。

```
127.0.0.1:6379>
shiyanshi@507:~/jiangyuehua/redis-4.0.10$ src/redis-
127.0.0.1:6379> auth 123456
OK
127.0.0.1:6379> rpush msg "hello world"
(integer) 1
127.0.0.1:6379> rpush msg "hello Logstash"
(integer) 1
127.0.0.1:6379> rpush msg "hello elasticsearch"
(integer) 1
127.0.0.1:6379>
```

```
{
      "@version" => "1",
    "@timestamp" => 2018-07-03T07:58:09.600Z,
       "message" => "hello Logstash"
}
{
      "@version" => "1",
    "@timestamp" => 2018-07-03T07:59:55.285Z,
       "message" => "hello elasticsearch"
}
```

图 6.9　redis 端的数据情况及基于列表的 Logstash 处理后的结果

2. 测试基于 pattern_channel 的日志数据流

基于消息订阅机制，在 redis 中也可使用 publish channel message 命令将信息发送到指定的频道。修改图 6.8 的代码，改为输入数据来源于 pattern_channel，并修改 key 为 msg *，其他的参数采用默认值或者类似于图 6.8 中的设置即可，如图 6.10 所示。

```
input {
    redis {
        codec => "plain"
        batch_count => 2
        data_type => "pattern_chanel"
        host => "127.0.0.1"
        key => "msg*"
        password => "123456"
    }
}
output{
    stdout {}
}
```

图 6.10　Logstash 的配置

然后启动 Logstash。在 redis 的客户端通过 publish 命令向名为 msg1、msg2、msg3 的 3 个频道发布信息，如图 6.11 左侧所示。此例中第一次发布的消息是字符串"China"，下方显示的数字 1 表示有一个订阅者(这个订阅者就是 Logstash，这个输入到 msg1 的信息已经由 Logstash 订阅并显示到屏幕，如图 6.11 右侧所示)。同理，可依次在 redis 端向不同的频道发布"Japan"、"USA"等字符串，它们分别经由 Logstash 这个管道分

别输出到 msg1 和 msg2 并显示在屏幕上。

```
127.0.0.1:6379> publish msg1 "China"
(integer) 1
127.0.0.1:6379> publish msg2 "Japan"
(integer) 1
127.0.0.1:6379> publish msg3 "USA"
(integer) 1
127.0.0.1:6379>
127.0.0.1:6379>
127.0.0.1:6379>
127.0.0.1:6379>
127.0.0.1:6379>
127.0.0.1:6379>
127.0.0.1:6379>
```

```
       "@version" => "1",
     "@timestamp" => 2018-07-03T08:21:03.462Z,
        "message" => "China"
}
{
       "@version" => "1",
     "@timestamp" => 2018-07-03T08:21:23.309Z,
        "message" => "Japan"
}
{
       "@version" => "1",
     "@timestamp" => 2018-07-03T08:21:38.040Z,
        "message" => "USA"
```

图 6.11 redis 端的数据情况及基于频道的 Logstash 处理后的结果

6.2.5 处理基于 TCP 传输的日志数据

Logstash 可以从 TCP Socket 中获取日志数据。像 stdin 和 file 方法一样，这里假定每一行都是一个日志。

> Tips TCP 是因特网中的传输层协议，使用三次握手协议建立连接。当主动方发出 syn 连接请求后，等待对方回答 syn+ack，并最终对对方的 syn 执行 ack 确认。

代码段 6.8 给出基于 TCP 传输到输入端的 Logstash 的配置文件格式。

代码段 6.8：Logstash 配置文件格式，处理基于 TCP 传输的日志数据

```
input {
  tcp {
    add_field=>…                #哈希值,可选项,默认为{}
    codec=>…                    #可选项,默认为 line
    host=>…                     #可选项, 默认为 0.0.0.0
    mode=>…                     #可选项,取值为"server"、"client"之一,默认为"server"
    port=>…                     #必选项,端口号,需和通信的另一端的端口号匹配
    ssl_cert=>…                 #一个可用的文件系统路径,可选项
    ssl_enable=>…               #布尔值,可选项,默认为 false
    ssl_key=>…                  #一个可用的文件系统路径,可选项
    ssl_key_passphrase=>…       #密码,可选项,默认为 nil
    ssl_verify=>…               #布尔值,可选项,默认为 true
    tags=>…                     #数组,可选项
    type=>…                     #字符串,可选项
  }
}
```

1. 将 Logstash 的 input 部分作为客户端

首先，设计 Java 应用程序并使之充当服务器端。用于发送信息的程序实现方法如代码段 6.9 所示。作为客户端的 Logstash 则基于 TCP，接收来自服务器端传来的信息并显示出来。

代码段 6.9：Java 应用程序（服务器端）

```
import java.io.*;
import java.net.ServerSocket;
import java.net.Socket;
public class TcpServer {
    public static void main(String[] args) throws IOException,
    InterruptedException {
        ServerSocket serverSocket=new ServerSocket(5656);       //设定端口号
        for(int i=0;i<=3;i++){                                //循环 4 次，结果如图 6.12 所示
            Socket socket=serverSocket.accept();              //阻塞等待消息
            OutputStream outputStream=socket.getOutputStream();
            outputStream.write(("Welcome, logstash"+i).getBytes());
            outputStream.close();
             Thread.sleep(999999);
            socket.close();
        }
        serverSocket.close();
    }
}
```

其次，配置 Logstash 端，如图 6.12 所示。注意，这里的参数 mode 用于设定 Logstash 作为客户端，port 指定的端口号要和代码段 6.9 中服务器端所开启的监听端口号一致。

```
input {
    tcp {
        host => "192.168.1.104"
        mode => "client"
        port => "5656"
    }
}
output{
    stdout {}
}
```

图 6.12　Logstash 端配置

最后，依次启动 Java 应用程序（即启动服务器端，如图 6.13 上图所示）、Logstash（即启动客户端，如图 6.13 下图所示）。按照代码段 6.9 中的设计思路，服务器端发送指定的字符串，可看到由服务器端发出的信息已经经过 Logstash 这个管道流向了屏幕（即客户端，如图 6.13 下图所示）。

2. 将 Logstash 的 input 部分作为服务器

首先，设计 Java 应用程序并使之作为客户端，如代码段 6.10 所示；将 Logstash 的 input 部分作为服务器模式。

```
import java.io.*;
import java.net.ServerSocket;
import java.net.Socket;

public class TcpServer {
    public static void main(String[] args) throws IOException, InterruptedException {
        ServerSocket serverSocket = new ServerSocket( port: 5656);         //设定端口号
        for (int i = 0; i <= 3; i++) {                                  //循环4次
            Socket socket = serverSocket.accept();//阻塞等待消息
            OutputStream outputStream = socket.getOutputStream();
            outputStream.write(("Welcome, logstash" + i).getBytes());
            outputStream.close();
            Thread.sleep( millis: 999999);
            socket.close();
        }
        serverSocket.close();
    }
}
```

```
{
       "message" => "Welcome, logstash1",
          "host" => "192.168.1.104",
          "port" => 5656,
    "@timestamp" => 2018-07-03T09:02:25.592Z,
      "@version" => "1"
}
{
       "message" => "Welcome, logstash2",
          "host" => "192.168.1.104",
          "port" => 5656,
    "@timestamp" => 2018-07-03T09:02:26.580Z,
      "@version" => "1"
}
{
       "message" => "Welcome, logstash3",
          "host" => "192.168.1.104",
          "port" => 5656,
    "@timestamp" => 2018-07-03T09:02:27.576Z,
      "@version" => "1"
}
```

图 6.13 服务器端设置(上)及客户端输出(下)

代码段 6.10：Java 客户端设计

```
import java.io.* ;
import java.net.InetAddress;
import java.net.Socket;
import java.net.UnknownHostException;
import java.io.DataOutputStream;
public class TcpClient {
    public static void main(String[] args) {
        DataOutputStream dos=null;
        BufferedReader brNet=null;
        BufferedReader brKey=null;
        Socket s=null;
        try {
```

```
            //建立 Socket
            s= new Socket(InetAddress.getByName("192.168.1.111"), 5656);
            InputStream ips= s.getInputStream();
            OutputStream ops= s.getOutputStream();
            brKey= new BufferedReader(new InputStreamReader(System.in));
            dos= new DataOutputStream(ops);
            brNet= new BufferedReader(new InputStreamReader(ips));
            while (true) {
                String strWord= brKey.readLine();
                dos.writeBytes(strWord+ System.getProperty("line.separator"));
                if (strWord.equalsIgnoreCase("quit"))
                    break;
                else
                    System.out.println(strWord);
            }
        } catch (UnknownHostException e) {
            e.printStackTrace();
        } catch (IOException e) {
            e.printStackTrace();
        }
//后略
```

其次，设计 Logstash 端的配置信息，如图 6.14 所示。注意，这里的 mode 参数是设定 Logstash 作为服务器，port 参数指定的端口号要和代码段 6.10 中的客户端端口设置相匹配。

```
input {
    tcp {
        host => "0.0.0.0"
        mode => "server"
        port => "5656"
    }
}
output{
    stdout {}
}
```

图 6.14　Logstash 端配置

最后，依次启动 Logstash（服务器端）和 Java 应用程序（客户端）。在客户端输入一些字符（如图 6.15 左侧所示），它们会经由 Logstash 这个管道输送到屏幕上显示，如图 6.15 右侧所示。

```
Debugger  Console
E:\tools\java1
Connected to t
hello
hello

", :ssl_enable=>"false"}
[2018-07-03T19:14:18,477][INFO ][logstash.pipeline        ] Pipel
 :thread=>"#<Thread:0x55007a0c run>"}
[2018-07-03T19:14:18,627][INFO ][logstash.agent           ] Pipel
{
      "@version" => "1",
       "message" => "hello\r",
          "host" => "192.168.1.104",
          "port" => 60463,
    "@timestamp" => 2018-07-03T11:16:15.053Z
}
```

图 6.15　运行结果

6.2.6　处理基于 UDP 传输的日志数据

和基于 TCP 的信息传输过程类似，Logstash 也可从 UDP Socket 中获取数据。代码段 6.11 给出在 Logstash 配置文件中配置基于 UDP 传输数据流的方法，图 6.16 给出了实际的 Logstash 配置。

> Tips　UDP 是用户数据报协议，是一个简单的面向数据报的运输层协议。由于 UDP 在传输数据报前不用在客户和服务器之间建立连接且没有超时重发等机制，故而传输速度相对较快。

代码段 6.11：基于 UDP 的 Logstash 配置文件格式

```
input {
  udp {
    add_field=>…                    #哈希值,可选项,默认为{}
    buffer_size=>…                  #数值,可选项,默认为 8192
    codec=>…                        #编解码器,可选项,默认为 plain
    host=>…                         #字符串,可选项,默认为 0.0.0.0
    port=>…                         #数值,必选项
    queue_size=>…                   #数值,可选项,默认为 2000
    tags=>…                         #数组,可选项
    type=>…                         #字符串,可选项
    workers=>…                      #数值,可选项,默认为 2
  }
}
```

为验证运行效果，这里设计了基于 Java 的应用程序，如代码段 6.12 所示。注意，Java 应用程序中的端口号应和 Logstash 配置文件中的端口号一致。依次启动 Logstash 和 Java 应用程序 UDPClient，实际运行效果如图 6.17 所示。从图中可看出，输入的信息

```
input {
    udp {
        host => "0.0.0.0"
        port => "5656"
    }
}
output{
    stdout{}
}
```

图 6.16　基于 UDP 的 Logstash 配置文件

（图 6.17 的左侧）以 UDP 的方式经由 Logstash 这个管道显示在屏幕上（图 6.17 的右侧）。

代码段 6.12：测试程序

```
import java.io.*;
import java.net.*;
class UDPClient {
    public static void main(String[] args) throws IOException {
        DatagramSocket client=new DatagramSocket();
        InetAddress addr=InetAddress.getByName("192.168.1.111");
        int port=5656;
        while (true) {
            byte[] send=new BufferedReader(new InputStreamReader(System.in)).readLine().getBytes
            ();
            DatagramPacket sendPacket
                    =new DatagramPacket(send, send.length, addr, port);
            client.send(sendPacket);
        }
    }
}
```

```
DPClient
E:\tools\java1
hello
world
Logstash
```

```
    "@timestamp" => 2018-07-03T11:45:15.979Z,
      "@version" => "1",
       "message" => "world",
          "host" => "192.168.1.104"
}
{
    "@timestamp" => 2018-07-03T11:45:20.911Z,
      "@version" => "1",
       "message" => "Logstash",
          "host" => "192.168.1.104"
```

图 6.17　基于 UDP 的信息输入处理

6.3 codec:格式化日志数据

在 Logstash 中,codec 部分可用于格式化日志数据。codecs 使得 Logstash 可以更方便地与其他自定义数据格式的产品共存,如 graphite(开源的存储图形化展示的组件)、fluent、netflow、collectd(守护进程,是一种收集系统性能和提供各种存储方式来存储不同值的机制),以及使用 msgpack、json 等通用数据格式的其他产品。

> Tips codec 有针对 avro、cef、cloudfront、cloudtrail、collectd、compress_spooler、dots、edn、edn_lines、es_bulk、fluent、graphite、gzip_lines、json、json_lines、line、msgpack、multiline、netflow、nmap、oldlogstashjson、plain、rubydebug、s3_plain 等多种数据来源的插件,可格式化相应的数据。

本节主要介绍对 json、plain 等格式的数据的用法。

6.3.1 json 格式

可利用 codec 机制解析 json 格式的日志数据(如 json 流式数据是以\n 来分隔的,可参考 json_lines 方式)。代码段 6.13 给出了 codec 可采用的字符集形式。

代码段 6.13:Logstach 配置文件格式,使用 codec

```
input {
  file {
    codec=>json {
      charset=>… #string,必须是 ["ASCII-8BIT", "Big5", "Big5-HKSCS", "Big5-UAO", "CP949", "Emacs-Mule", "EUC-JP", "EUC-KR", "EUC-TW", "GB18030", "GBK", "ISO-8859-1", "ISO-8859-2", "ISO-8859-3", "ISO-8859-4", "ISO-8859-5", "ISO-8859-6", "ISO-8859-7", "ISO-8859-8", "ISO-8859-9", "ISO-8859-10", "ISO-8859-11", "ISO-8859-13", "ISO-8859-14", "ISO-8859-15", "ISO-8859-16", "KOI8-R", "KOI8-U", "Shift_JIS", "US-ASCII", "UTF-8", "UTF-16BE", "UTF-16LE", "UTF-32BE", "UTF-32LE", "Windows-1251", "GB2312", "IBM437", "IBM737", "IBM775", "CP850", "IBM852", "CP852", "IBM855", "CP855", "IBM857", "IBM860", "IBM861", "IBM862", "IBM863", "IBM864", "IBM865", "IBM866", "IBM869", "Windows-1258", "GB1988", "macCentEuro", "macCroatian", "macCyrillic", "macGreek", "macIceland", "macRoman", "macRomania", "macThai", "macTurkish", "macUkraine", "CP950", "CP951", "stateless-ISO-2022-JP", "eucJP-ms", "CP51932", "GB12345", "ISO-2022-JP", "ISO-2022-JP-2", "CP50220", "CP50221", "Windows-1252", "Windows-1250",
```

```
"Windows-1256", "Windows-1253", "Windows-1255", "Windows-1254", "TIS-620", "Windows-874", "Windows-1257", "Windows-31J", "MacJapanese", "UTF-7", "UTF8-MAC", "UTF-16", "UTF-32", "UTF8-DoCoMo", "SJIS-DoCoMo", "UTF8-KDDI", "SJIS-KDDI", "ISO-2022-JP-KDDI", "stateless-ISO-2022-JP-KDDI", "UTF8-SoftBank", "SJIS-SoftBank", "BINARY", "CP437", "CP737", "CP775", "IBM850", "CP857", "CP860", "CP861", "CP862", "CP863", "CP864", "CP865", "CP866", "CP869", "CP1258", "Big5-HKSCS:2008", "eucJP", "euc-jp-ms", "eucKR", "eucTW", "EUC-CN", "eucCN", "CP936", "ISO2022-JP", "ISO2022-JP2", "ISO8859-1", "CP1252", "ISO8859-2", "CP1250", "ISO8859-3", "ISO8859-4", "ISO8859-5", "ISO8859-6", "CP1256", "ISO8859-7", "CP1253", "ISO8859-8", "CP1255", "ISO8859-9", "CP1254", "ISO8859-10", "ISO8859-11", "CP874", "ISO8859-13", "CP1257", "ISO8859-14", "ISO8859-15", "ISO8859-16", "CP878", "CP932", "csWindows31J", "SJIS", "PCK", "MacJapan", "ASCII", "ANSI_X3.4-1968", "646", "CP65000", "CP65001", "UTF-8-MAC", "UTF-8-HFS", "UCS-2BE", "UCS-4BE", "UCS-4LE", "CP1251", "external", "locale"]其中之一 (可选)，默认："UTF-8"
      }
    }
}
```

下面给出 codec 和 output 一起使用的例子。图 6.18 所示的 Logstash 配置文件在 output 中指定的数据格式为 json。此后运行 Logstash，则用户输入的字符串会以 json 格式显示。图 6.19 为分别输入了 helloworld 字符串和 this is an example 字符串后的情况，可以看出，输入的字符串都被解析为 json 格式的数据了。

```
input {
    stdin{}
}
output {
    stdout{codec => json}
}
```

图 6.18 指定输出格式为 json 的 Logstash 配置文件

如果采用 json_lines，则可解析按行分隔的 json 格式

```
shiyanshi@507:~/jiangyuehua/logstash-6.2.4/bin$ ./logstash -f conf6.6
Sending Logstash's logs to /home/shiyanshi/jiangyuehua/logstash-6.2.4/logs which is now configured via log4j
[2018-07-03T19:54:41,280][INFO ][logstash.modules.scaffold] Initializing module {:module_name=>"netflow", :d
on"}
[2018-07-03T19:54:41,316][INFO ][logstash.modules.scaffold] Initializing module {:module_name=>"fb_apache",
ration"}
[2018-07-03T19:54:42,209][WARN ][logstash.config.source.multilocal] Ignoring the 'pipelines.yml' file becaus
[2018-07-03T19:54:43,522][INFO ][logstash.runner          ] Starting Logstash {"logstash.version"=>"6.2.4"}
[2018-07-03T19:54:44,525][INFO ][logstash.agent           ] Successfully started Logstash API endpoint {:por
[2018-07-03T19:54:48,170][INFO ][logstash.pipeline        ] Starting pipeline {:pipeline_id=>"main", "pipeli
[2018-07-03T19:54:48,507][INFO ][logstash.pipeline        ] Pipeline started successfully {:pipeline_id=>"ma
The stdin plugin is now waiting for input:
[2018-07-03T19:54:48,654][INFO ][logstash.agent           ] Pipelines running {:count=>1, :pipelines=>["main
helloworld
{"@version":"1","host":"507","message":"helloworld","@timestamp":"2018-07-03T11:55:28.254Z"}this is example
{"@version":"1","host":"507","message":"this is example","@timestamp":"2018-07-03T11:55:47.721Z"}
```

图 6.19 输出 json 格式的数据

的日志数据。代码段 6.14 是 Logstash 配置文件，它给出了在 UDP 输入模式下的 json_lines 的用法。

代码段 6.14：在 UDP 输入模式中采用 codec 的配置文件

```
input {
  udp {
    port=>5656
    codec=>json_lines {
      harset=>…                    #字符串,可选项,默认为"UTF-8"
    }
  }
}
```

6.3.2 rubydebug 格式

rubydebug 解析器使用 ruby awesome print 库来解析日志格式。图 6.20 给出了 Logstash 的配置信息，注意，这里设置的输入信息是 json 格式的（因此输入信息时应采用 json 格式），而数据输出时则采用 rubydebug 格式。图 6.21 是实际运行结果，其中第 1 行是输入的 json 格式的数据，而从第 3 行开始输出的是 rubydebug 格式的信息。

```
input {
    stdin{codec => json}
}
output {
    stdout{codec => rubydebug}
}
```

图 6.20 指定输出格式为 rubydebug 的 Logstash 配置文件

```
{"name": "xiaoming", "age": 18}
{
          "host" => "507",
      "@version" => "1",
           "age" => 18,
          "name" => "xiaoming",
    "@timestamp" => 2018-07-03T12:45:47.816Z
}
```

图 6.21 输出 rubydebug 格式的数据

6.3.3 plain 格式

plain 是一个空的解析器，其解析格式可以由用户自行指定。代码段 6.15 给出当 Logstash 输入源是文件的情况下通过 codec 进行格式转换的方法。图 6.22 给出了实际结果，其中，输出字符串的部分不带任何格式。

代码段 6.15：Logstash 输出采用 plain 格式

```
input {
    stdin{ }
}
output {
    stdout{
        codec=>"plain"
    }
}
```

```
[2018-07-03T20:11:16,125][INFO ][logstash.modules.scaffol
ration"}
[2018-07-03T20:11:17,186][WARN ][logstash.config.source.m
[2018-07-03T20:11:18,406][INFO ][logstash.runner
[2018-07-03T20:11:19,554][INFO ][logstash.agent
[2018-07-03T20:11:22,553][INFO ][logstash.pipeline
[2018-07-03T20:11:22,766][INFO ][logstash.pipeline
The stdin plugin is now waiting for input:
[2018-07-03T20:11:22,930][INFO ][logstash.agent
hello
2018-07-03T12:11:35.759Z 507 hello
```

图 6.22　输出 plain 格式的数据

6.4　基于 filter 的日志处理与转换

在基于 Logstash 的日志处理中，往往要处理多种不同的日志信息，如 Apache 服务器日志、Postfix 日志（Wietse Venema 在 IBM 公司的 GPL 协议下开发的一种邮件传输代理软件）、Java 应用程序或与某种特定应用相关的日志信息等（Turnbull，2015）。前面已经介绍了如何让 Logstash 直接处理各种来源的日志数据，但尚未利用、抽取、过滤这些数据中可能蕴含的元数据信息。例如，以下的 Apache 服务器日志信息包含了 IP 地址、时间戳、HTTP 方法、HTTP 响应模式等诸多信息。

```
186.4.131.228- - [20/Dec/2012:20:34:08 - 0500] "GET /2012/12/new- product/ HTTP/1.0"200 10902 "http://www.example.com/20012/12/new- product/""Mozilla/5.0 (Windows; U; Windows NT 5.1; pl; rv:1.9.1.3) Gecko/20090824 Firefox/3.5.3"
```

对上述日志来说，如果不做任何解析处理，而是直接将这些信息交给 Logstash，那么即使经过 Logstash 的处理，用户可能也不清楚其中的各个部分的含义。利用 Logstash 提供的 filter 机制可以方便地按照用户要求解析日志信息，如果需要抽取、分析、计算可能蕴含

的元数据信息，可以利用 Logstash 中的 filter 机制完成相应的功能。

> Tips filter 有针对 aggregate、alter、anonymize、cidr、cipher、clone、collate、csv、date、de _ dot、dissect、dns、drop、elapsed、elasticsearch、environment、extractnumbers、fingerprint、geoip、grok、i18n、json、json _ encode、kv、metaevent、metricize、metrics、mutate、oui、prune、punct、range、ruby、sleep、split、syslog _ pri、throttle、tld、translate、urldecode、useragent、uuid、xml、yaml、zeromq 等多种来源数据的插件，可过滤和分析相应的数据。

限于篇幅，本章只对其中的 json filter、grok filter、kv filter 进行介绍。

6.4.1 json filter

如果日志数据源是 json 格式的，则利用 filter，可将其扩展成一个编程人员所需要的数据结构。代码段 6.16 给出在 filter 中使用 json 格式数据的方法。

代码段 6.16：Logstash 配置文件格式，用于解析并处理 json 格式的数据为指定的数据结构

```
filter {
  json {
    add_field=>…                    #哈希值,可选项,默认为空
    skip_on_invalid_json=>…         #布尔值,可选项,默认为 false
    add_tag=>…                      #数组,可选项, 默认为空
     periodic_flush=>…              #布尔值,可选项, 默认为 false
    remove_field=>…                 #数组,可选项, 默认为空
    remove_tag=>…                   #数组,可选项, 默认为空
    source=>…                       #字符串,必选项
    target=>…                       #字符串,可选项
  }
}
```

对代码段 6.16 中的部分内容解释如下：

- add_field：值类型为哈希值，默认为空。设置后就会增加指定的字段到这个事件中。
- add_tag：值类型为数组，默认为空。若执行成功，会增加一个标签。
- periodic_flush：值类型为布尔值，默认为 false。定期调用过滤器的清缓存方法。
- remove_field：值类型为数组，默认为空。若执行成功，则删除一个字段。
- source：值类型为字符串，默认为未设置。
- skip_on_invalid_json：值类型为布尔值，默认为 false，跳过无效的 Json 数据，不

警告。

代码段 6.17 给出 Logstash 的配置信息。在 input、output 部分中间的 filter 部分嵌入了对 json 数据的解析方法，其中的 source=>是指要让 json filter 插件解析指定字段的 json 格式数据。例如，source=>"message"是让 json filter 解析 message 字段的数据。图 6.23给出了实际的运行结果。当用户输入 json 格式的数据后，Logstash 的 filter 会对输入信息进行加工处理。在此例中，用户输入的字符串是{"name"："lily"，"age"：13}，经处理后返回的结果显示在图 6.23 下方。

代码段 6.17：基于 json filter 的 Logstash 配置文件

```
input {
    stdin{ }
}
filter {
    json {
      source=>"message"
    }
}
output {
   stdout { codec=>"rubydebug" }
}
```

```
{"name": "lily", "age": 13}
{
       "message" => "{\"name\": \"lily\", \"age\": 13}",
      "@version" => "1",
          "host" => "507",
    "@timestamp" => 2018-07-03T12:49:59.995Z,
          "name" => "lily",
           "age" => 13
}
```

图 6.23　对 json 格式数据的 filter 处理结果

6.4.2　grok filter

grok 是一个数据结构化工具。利用它，只需要通过简单定义，就可将文本格式的字符串转换为结构化的数据。Logstash 默认带有上百个 grok 变量，可以直接使用，或者稍加改动，变成自定义的 grok。grok filter 适合对 syslog、Apache log 等可读日志的解析。代码段 6.18给出基于 grok filter 的 Logstash 配置文件格式，其中的 add_field 等的含义同上，不再赘述。

代码段 6.18：基于 grok filter 的 Logstash 配置文件格式

```
filter {
  grok {
    add_field=>…
    add_tag=>…
    break_on_match=>…              #布尔值,可选项,默认为 true
    keep_empty_captures=>…         #布尔值,可选项,默认为 false
    match=>…                       #哈希值,可选项,默认为空
    named_captures_only=>…         #布尔值,可选项,默认为 true
    overwrite=>…                   #数组,可选项,默认为空
    patterns_dir=>…                #数组,可选项,默认为空
    patterns_files_glob=>…         #字符串,可选项,默认为 *
    periodic_flush=>…
    remove_field=>…                #数组,可选项,默认为空
    remove_tag=>…
    tag_on_failure=>…              #数组,可选项,默认为 ["_grokparsefailure"]
    tag_on_timeout=>…              #字符串,可选项,默认为_groktimeout
    timeout_millis=>…              #数值,可选项,默认为 2000
  }
}
```

代码段 6.19 给出了一个基于 grok filter 的 Logstash 配置实例。注意如下代码：

```
match=>{ "message"=>%{TIMESTAMP_ISO8601:timestamp}  %{LOGLEVEL:log-level}
                    \[%{DATA:class}\] %{GREEDYDATA:message}" }
```

其中，match 是一个哈希类型的集合。其键部分指定了需要匹配的字段，如本例中需要匹配 message 字段的数据；其后的值部分则使用正则匹配字符串并为它们分别指定一个名字，本例包括如下几项：

- 按 TIMESTAMP_ISO8601 正则项匹配文本并指定其变量名为 timestamp。
- 按 LOGLEVEL 正则项匹配文本并指定其变量名为 log-level。
- 按 DATA 正则项匹配文本并指定其变量名为 class。
- 按 GREEDYDATA 正则项匹配文本并指定其变量名为 message。

> Tips GitHub 提供了大量的已经定义好的模式，用户可以轻松组合各种模式，满足不同场景和特殊的需求。可从 GitHub 的官方网站查阅具体信息，链接地址为 https://github.com/logstash-plugins/logstash-patterns-core/tree/master/patterns。

在此基础上，Logstash 可对用户输入的内容进行解析。

代码段 6.19：基于 grok filter 的 Logstash 配置实例

```
input {
  stdin{ }
}
filter {
  grok {
    match=> { "message"=> "%{TIMESTAMP_ISO8601:timestamp}  %{LOGLEVEL:log-level}
                          \[%{DATA:class}\] %{GREEDYDATA:message}" ]
  }
}
output {
  stdout { codec=> rubydebug }
}
```

Tips 由于正则匹配的模糊性，如果开始就将所有正则项写好，一旦其中有一个变量不能匹配到正确文本，就可能会带来调试上的麻烦。所以，推荐先使用贪婪策略匹配整个字符串，然后逐个添加需要匹配的字段，例如：

```
%{GREEDYDATA:message}                                          #先匹配整个字符串
%{TIMESTAMP_ISO8601:timestamp} %{GREEDYDATA:message}           #再尝试添加匹配项
```

例如，当处理“2018-07-03T23:56:42.000＋00:00　INFO［servlet.mvc.annotation.DefaultAnnotationHandlerMapping:315］Mapped URL path［/system/negative/index］onto handler 'negativeAction'”日志信息时，按照代码段 6.19 中的 Logstash 配置文件 conf.conf 进行解析，会得到如图 6.24 所示的结果（大括号中的内容是解析后的结果）。

```
2018-07-03T23:56:42.000+00:00  INFO [servlet.mvc.annot
n'
{
      "timestamp" => "2018-07-03T23:56:42.000+00:00",
      "log-level" => "INFO",
          "class" => "servlet.mvc.annotation.DefaultAnn
       "@version" => "1",
        "message" => [
        [0] "2018-07-03T23:56:42.000+00:00  INFO [serv
negativeAction'",
        [1] "Mapped URL path [/system/negative/index]
    ],
     "@timestamp" => 2018-07-05T03:16:28.132Z,
           "host" => "507"
}
```

图 6.24　基于 grok filter 的处理结果

6.4.3　kv filter

kv filter 用于解析键值对数据。代码段 6.20 给出基于 kv filter 的 Logstash 配置。其

中，field_split 用来设置分隔符，代码 kv {field_split=>"&?"}的含义是说明字符串分隔符是 & 或者?。例如，在图 6.25 中，用户输入的日志字符串是 http://www.baidu.com/s?wd=%E4%B8%AD%E5%9B%BD&rsv_spt=1&issp=1&f=8&rsv_bp=0&rsv_idx=2&ie=utf-8&tn=SE_hldp01550_7zn76813&rsv_enter=1&rsv_sug3=2&rsv_sug1=1&rsv_sug2=0&inputT=2950&rsv_sug4=2951&rsv_sug=2，经过 Logstash 的处理后，得到以 & 或?分隔的子字串。

代码段 6.20：基于 kv filter 的 Logstash 配置文件

```
input {
    stdin { }
}
filter {
    kv {
        field_split=>"&?"
    }
}
output {
    stdout {
        codec=>rubydebug
    }
}
```

```
http://www.baidu.com/s?wd=%E4%B8%AD%E5%9B%BD&rsv_spt=1&issp
nputT=2950&rsv_sug4=2951&rsv_sug=2
{
            "tn" => "SE_hldp01550_7zn76813",
      "rsv_sug4" => "2951",
        "inputT" => "2950",
      "rsv_sug1" => "1",
    "@timestamp" => 2018-07-05T04:15:12.869Z,
      "@version" => "1",
            "wd" => "%E4%B8%AD%E5%9B%BD",
       "message" => "http://www.baidu.com/s?wd=%E4%B8%AD%E5
v_sug1=1&rsv_sug2=0&inputT=2950&rsv_sug4=2951&rsv_sug=2",
     "rsv_enter" => "1",
      "rsv_sug2" => "0",
          "issp" => "1",
        "rsv_bp" => "0",
             "f" => "8",
          "host" => "507",
            "ie" => "utf-8",
      "rsv_sug3" => "2",
       "rsv_sug" => "2",
       "rsv_spt" => "1",
       "rsv_idx" => "2"
}
```

图 6.25 基于 kv filter 的处理结果

从图 6.25 中还可以看出，解析出的查询字符串 wd 是 URL 字符串模式（为%E4%

B8%AD%E5%9B%BD)，而并非用户在键盘输入的可以识别的中文字符串(应该是“中国”二字)。为解决上述问题，只需要在配置文件中添加 urldecode 部分即可，详见代码段 6.21，基于该配置文件的 Logstash 处理结果如图 6.26 所示。可以清楚地看到，图 6.26 已经顺利解析出了 wd 中隐含的内容。

代码段 6.21：基于 kv filter 和 urldecode 的 Logstash 配置文件

```
input {
    stdin{ }
}
filter {
    kv {
        field_split=>"&?"            #说明字符串分隔符是 & 或者？
    }
    urldecode {
        field=>wd                    #说明解码字段是 wd
    }
}
output {
    stdout {
        codec=>rubydebug
    }
}
```

```
http://www.baidu.com/s?wd=%E4%B8%AD%E5%9B%BD&rsv_spt=1&issp=1&f=8&rs
v_bp=0&rsv_idx=2&ie=utf-8&tn=SE_hldp01550_7zn76813&rsv_enter=1&rsv_s
ug3=2&rsv_sug1=1&rsv_sug2=0&inputT=2950&rsv_sug4=2951&rsv_sug=2
{
            "wd" => "中国",
      "rsv_sug2" => "0",
            "tn" => "SE_hldp01550_7zn76813",
    "@timestamp" => 2018-07-05T05:41:29.471Z,
             "f" => "8",
          "host" => "507",
      "rsv_sug1" => "1",
          "issp" => "1",
       "rsv_idx" => "2",
        "rsv_bp" => "0",
            "ie" => "utf-8",
      "@version" => "1",
      "rsv_sug3" => "2",
       "message" => "http://www.baidu.com/s?wd=%E4%B8%AD%E5%9B%BD&rs
v_spt=1&issp=1&f=8&rsv_bp=0&rsv_idx=2&ie=utf-8&tn=SE_hldp01550_7zn76
813&rsv_enter=1&rsv_sug3=2&rsv_sug1=1&rsv_sug2=0&inputT=2950&rsv_sug
4=2951&rsv_sug=2",
       "rsv_spt" => "1",
     "rsv_enter" => "1",
        "inputT" => "2950",
      "rsv_sug4" => "2951",
       "rsv_sug" => "2"
}
```

图 6.26　基于 kv filter 和 urldecode 的处理结果

> Tips urldecode 是对字符串进行解码,其返回值是已解码的字符串。其解码规则一般是:数字和字母不变,空格变为+,其他字符被解码,例如"中国"二字的解码形式为%E4%B8%AD%E5%9B%BD(在每个字节前加%)。

6.5 output:输出日志数据

前面已经对 Logstash 中的 input、codec、filter 进行了介绍。在 Logstash 配置文件的 output 部分,可以使用 stdout 把经由 Logstash 处理的日志传送到显示器上输出。当然,在设置输出时,也可以在 stdout 中同时设置 codec,如代码段 6.21 所示。其实,当 Logstash 处理完日志数据后,不仅可以在显示器上显示,也可根据需要使用 elasticsearch、email、redis、file 等插件来完成输出。

> Tips Logstash 可以将处理之后的数据使用 boundary、circonus、cloudwatch、csv、datadog、datadog_metrics、elasticsearch、email、exec、file、ganglia、gelf、google_bigquery、google_cloud_storage、graphite、graphtastic、hipchat、http、influxdb、irc、jira、juggernaut、kafka、librato、loggly、lumberjack、metriccatcher、mongodb、nagios、nagios_nsca、newrelic、opentsdb、pagerduty、pipe、rabbitmq、rackspace、redis、redmine、riak、riemann、s3、sns、solr_http、sqs、statsd、stdout、stomp、syslog、tcp、udp、webhdfs、websocket、xmpp、zabbix、zeromq 等插件进行输出。

本节介绍输出日志数据文件的方法。

6.5.1 将处理后的日志输出到 Elasticsearch 中

通过设置 Logstash 的 output 部分中的 elasticsearch 部分,可将 Elasticsearch 作为处理日志的接收端。这种方式可以将收集到的日志通过 HTTP 接口存放到 Elasticsearch 中。代码段 6.22 给出 Logstash 配置文件中的 output 部分的设置。

代码段 6.22:Logstash 配置文件中的 output 部分的格式

```
output {
  elasticsearch {
    codec=>…                    #编解码器,可选项,默认为 plain
```

```
        document_id=>…                   #字符串,可选项
        flush_size=>…                    #数值,可选项,默认为 500
        hosts=>…                         #数组,可选项,Elasticsearch服务器的主机列表,默认
                                         #为数组["127.0.0.1"]
        idle_flush_time=>…               #数值,可选项,默认为 1
        index=>…                         #字符串,可选项,默认为 "logstash-%{+YYYY.MM.dd}"
        manage_template=>…               #布尔值,可选项,默认为 true
        password=>…                      #密码,可选项
        template=>…                      #有效的文件路径,可选项
        template_name=>…                 #字符串,可选项,默认为 logstash
        template_overwrite=>…            #布尔值,可选项,默认为 false
        user=>…                          #字符串,可选项
    }
}
```

代码段 6.23 给出了一个完整的 Logstash 配置文件实例，实际运行结果如图 6.27 所示。

代码段 6.23：Logstash 配置文件，用 Elasticsearch 作为日志接收端

```
input {
    stdin{ }
}
output {
    elasticsearch {                  #通过 HTTP 方式将数据传输到 Elasticsearch 中
        hosts=>["localhost"]         #指定数组形式的主机列表
    }
    stdout{}
}
```

然后，依次启动 Elasticsearch、Logstash，可在控制台输入部分字符。根据代码段 6.23 中的设置，输入的信息既可显示在屏幕上，也会存入 Elasticsearch 中。图 6.27 是将数据存入 Elasticsearch 索引文件后的结果。

查询 5 个分片中用的 5 个. 552 命中. 耗时 0.011 秒

_index	_type	_id	_score ▲	message
logstash-2018.07.02	doc	5EWmWWQBmC1EBcTh6GCM	1	Copyright 2012-2018 Elasticsearch
logstash-2018.07.02	doc	7EWmWWQBmC1EBcTh6GCM	1	==
logstash-2018.07.02	doc	70WmWWQBmC1EBcTh6GCM	1	the following conditions:
logstash-2018.07.02	doc	9kWmWWQBmC1EBcTh6GCM	1	included in all copies or substantial portions of the Software.

图 6.27　存入 Elasticsearch 索引文件的信息

6.5.2 将处理后的日志输出到文件中

Logstash 也可以将收集到的日志(经处理后)输出到一个指定的文件中(可以用域中的一些值作为文件名或者路径名)。在代码段 6.24 的 Logstash 配置文件中指定用文件作为日志接收端。

代码段 6.24: Logstash 配置文件格式,用文件作为日志接收端

```
output {
    file {
        codec=>…                    #编解码器,可选项,默认为 json_lines
        flush_interval=>…           #数组,可选项,默认为 2
        gzip=>…                     #布尔值,可选项,默认为 false
        path=>…                     #字符串,必选项,待存放文件的路径
        workers=>…                  #数值,可选项,默认为 1
    }
}
```

作为实例,代码段 6.25 给出一个完整的 Logstash 配置文件。其中,field_split 这个参数用来设置分隔符(由于这里给出的日志字符串是用户输入的 URL,因此指定 & 或?作为分隔符),urldecode 用于对字符串进行 URL 解码。URL 字符串经过 Logstash 处理后,存入指定的文件中,如图 6.28 所示。

代码段 6.25: 用指定路径的文件来存放处理后的日志信息

```
input {
  stdin { }
}
filter {
  kv {
    field_split=>"&?"
  }
  urldecode {
      field=>wd
  }
}
output {
  file {
    path=>"/log.txt"
    codec=>json
  }
```

```
    stdout {
      codec=>plain
    }
}
```

```
shiyanshi@507:~/jiangyuehua/logstash-6.2.4/bin$ cat log.txt
{"rsv_bp":"0","@version":"1","ie":"utf-8","rsv_sug4":"2951","issp"
:"1","rsv_sug":"2","host":"507","rsv_spt":"1","tn":"SE_hldp01550_7
zn76813","rsv_sug2":"0","message":"http://www.baidu.com/s?wd=%E4%B
8%AD%E5%9B%BD&rsv_spt=1&issp=1&f=8&rsv_bp=0&rsv_idx=2&ie=utf-8&tn=
SE_hldp01550_7zn76813&rsv_enter=1&rsv_sug3=2&rsv_sug1=1&rsv_sug2=0
&inputT=2950&rsv_sug4=2951&rsv_sug=2","f":"8","rsv_enter":"1","rsv
_sug1":"1","rsv_sug3":"2","@timestamp":"2018-07-05T06:10:33.851Z",
"wd":"中国","rsv_idx":"2","inputT":"2950"}
```

图 6.28 存入文件中的处理后的日志信息

6.5.3 将处理后的日志输出到 csv 文件中

通过对启动配置文件的设置，也可以将收集到的日志以 csv 的格式输出到指定的文件中。代码段 6.26 给出 Logstash 配置文件中有关使用 csv 插件输出的部分。

代码段 6.26：Logstash 配置文件格式，用 csv 作为日志接收端

```
output {
  csv {
    codec=>…                 #编解码器,可选项,默认为 plain
    csv_options=>…           #哈希值,可选项,默认为空
    fields=>…                #数组,必选项,指定 csv 中各个域的名称
    flush_interval=>…        #数值,可选项,默认为 2
    gzip=>…                  #布尔值,可选项,默认为 false
    path=>…                  #字符串,必选项,指定输出的 csv 文件路径
    workers=>…               #数值,可选项,默认为 1
  }
}
```

代码段 6.27 给出了一个完整的 Logstash 配置文件，从中可以看出，在对输入的日志字符串进行 kv filter、urldecode 等处理后，将解析的部分字段结果（如@timestamp、host、wd 等部分）存入指定的文件中，结果如图 6.29 所示。

代码段 6.27：Logstash 配置文件，用 csv 作为日志接收端

```
input {
  stdin { }
}
```

```
filter {
  kv {
    field_split=>"&?"
  }
  urldecode    {
      field=>wd
  }
}
output {
  csv{
    path=>"/log.txt"
    fields=>["@timestamp","host","wd"]
  }
  stdout {
   codec=>plain
  }
}
```

```
shiyanshi@507:~/jiangyuehua/logstash-6.2.4/bin$ cat log.csv
2018-07-05T06:17:21.040Z,507,中国
```

图 6.29 存入 csv 文件中的处理后的部分日志信息

6.5.4 将处理后的日志输出到 redis 中

Logstash 可以将处理后的日志输出到 redis 中。通过在 Logstash 配置文件的 output 部分使用 redis 插件，可以将处理之后的日志信息输出到 redis 中。代码段 6.28 给出 Logstash 配置文件的 output 中 redis 插件部分的实现方法。

代码段 6.28：Logstash 配置文件格式，有关 output 中 redis 的部分

```
output {
  redis {
    batch=>…                        #布尔值,可选项,默认为 false
    batch_events=>…                 #数值,可选项,默认为 50
    batch_timeout=>…                #数值,可选项,默认为 5
    codec=>…                        #编解码器,可选项,默认为 plain
    congestion_interval=>…          #数值,可选项,默认为 1
    congestion_threshold=>…         #数值,可选项,默认为 0
    data_type=>…                    #字符串,取值为"list"、"channel"之一,可选项
```

```
        db=>…                          #数值,可选项,默认为 0
        host=>…                        #数组,可选项,默认为 ["127.0.0.1"]
        key=>…                         #字符串,可选项
        password=>…                    #布尔值,可选项
        port=>…                        #数值,可选项,默认为 6379
        reconnect_interval=>…          #数值,可选项,默认为 1
        shuffle_hosts=>…               #布尔值,可选项,默认为 true
        timeout=>…                     #数值,可选项,默认为 5
        workers=>…                     #数值,可选项,默认为 1
    }
}
```

代码段 6.29 给出 Logstash 配置文件中 data_type 基于列表的实现。

代码段 6.29：Logstash 配置文件

```
input {
    stdin { }
}
output {
    redis {
        data_type=>"list"
        host=>["127.0.0.1"]
        key=>"example1"                #给定的列表名称,这里是一个示例字符串
        password=>123456               #对应于 redis 的密码
    }
}
```

依次启动 redis 的服务器端和客户端。基于代码段 6.29 的配置文件启动 Logstash。在 Logstash 控制台输入测试字符串，切换到 redis 的客户端，输入 lrange example1 0 -1（注意，这里的示例字符串 example1 要和代码段 6.29 中的 key 值匹配）。

Tips redis 中的语句 lrange key ＜start＞ ＜stop＞的作用是返回列表 key 中在指定区间内的元素，区间以偏移量 start 和 stop 指定，参数 start 和 stop 都从 0 开始，0 表示列表的第 1 个元素，1 表示列表的第 2 个元素，−1 表示列表的最后一个元素，−2 表示列表的倒数第二个元素，以此类推。

代码段 6.29 给出 Logstash 配置文件中基于列表的完整例子。和在 input 中的情况类

似，如果将这里的 data_type=> "list"改成 data_type=> "channel"，则输出信息会以 channel 的方式注入 redis 中。图 6.30 给出了基于 channel 的 redis 输出结果。注意，在测试时要用 subscribe 命令，且其后的 channel 名称应和在 Logstash 中定义的一致(例子中使用的 channel 名称为 example2，如图 6.30 左侧所示)。

Tips redis 中的语句 subscribe channel [channel …]用于订阅给定的一个或多个频道的信息。例如输入 subscribe msg chat_room，可能会返回如下信息：

```
1) "subscribe"          #返回值的类型,表示订阅成功
2) "msg"                #订阅的频道名字
3) (integer) 1          #目前已订阅的频道数量
```

```
shiyanshi@507:~/jiangyuehua/redis-4.0.10$ src/red
127.0.0.1:6379> auth 123456
OK
127.0.0.1:6379> subscribe example2
Reading messages... (press Ctrl-C to quit)
1) "subscribe"
2) "example2"
3) (integer) 1
1) "message"
2) "example2"
3) "{\"@version\":\"1\",\"message\":\"hello redis
```

```
"}
[2018-07-05T15:56:42,116][INFO ][logstash.modules.scaffold] Initializing module {:
tion"}
[2018-07-05T15:56:43,011][WARN ][logstash.config.source.multilocal] Ignoring the '
[2018-07-05T15:56:44,410][INFO ][logstash.runner          ] Starting Logstash {"l
[2018-07-05T15:56:45,445][INFO ][logstash.agent           ] Successfully started L
[2018-07-05T15:56:49,245][INFO ][logstash.pipeline        ] Starting pipeline {:pi
[2018-07-05T15:56:49,524][INFO ][logstash.pipeline        ] Pipeline started succe
The stdin plugin is now waiting for input:
[2018-07-05T15:56:49,727][INFO ][logstash.agent           ] Pipelines running {:co
hello redis
```

图 6.30 日志以 channel 的方式输出到 redis 中(左为 redis 输入端，右为 Logstash 输出端)

6.5.5 将处理后的日志通过 UDP 输出

在 Logstash 配置文件的 output 部分可以使用 UDP 的方式将日志通过网络发送到另外一台主机上。代码段 6.30 给出部分基于 UDP 方式输出文件的 Logstash 配置文件格式。

代码段 6.30：Logstash 配置文件格式

```
output {
  udp {
    codec=>…                #编解码器,可选项,默认为 json
    host=>…                 #字符串,必选项
    port=>…                 #数值,必选项
    workers=>…              #数值,可选项, 默认为 1
  }
}
```

代码段 6.31 给出了完整的 Logstash 配置文件内容。

代码段 6.31：Logstash 配置文件

```
input {
  stdin { }
}
output {
  udp {
    host=>"127.0.0.1"
    port=>5656
  }
}
```

为测试效果，代码段 6.32 给出 UDPClient（即传输数据的接收端）的 Java 实现。

代码段 6.32：UDPClient

```
import java.io.*;
import java.net.*;
class UDPClient {
    public static void main(String[] args) throws IOException {
        InetAddress addr=InetAddress.getByName("localhost");
        int port=5656;              //这个端口号要和 Logstash 配置文件中的端口号一致
        DatagramSocket client=new DatagramSocket(port,addr);
        while (true) {
            byte[] recvBuf=new byte[100];
            DatagramPacket recvPacket=new DatagramPacket(recvBuf, recvBuf.
            length);
            client.receive(recvPacket);
            String recvStr=new String(recvPacket.getData(), 0,recvPacket.
            getLength());
            System.out.println("收到:"+recvStr);
        }
    }
}
```

实际运行结果如图 6.31 所示。图 6.31 的上部是输入的信息，经过 Logstash 这个基于 UDP 的输出管道后，处理之后的信息如图 6.31 的下部所示。

6.5.6 将处理后的日志通过 TCP 输出

Logstash 可以将处理后的日志从 TCP Socket 中输出。代码段 6.33 给出了 Logstash 的 output 部分中有关 TCP 的设置。

```
shiyanshi@507:~/jiangyuehua/logstash-6.2.4/bin$ ./logstash -f ./conf.conf
Sending Logstash's logs to /home/shiyanshi/jiangyuehua/logstash-6.2.4/logs which is now configured via log4j2.prop
[2018-07-05T16:09:26,506][INFO ][logstash.modules.scaffold] Initializing module {:module_name=>"netflow", :directo
es/netflow/configuration"}
[2018-07-05T16:09:26,539][INFO ][logstash.modules.scaffold] Initializing module {:module_name=>"fb_apache", :direc
ules/fb_apache/configuration"}
[2018-07-05T16:09:27,547][WARN ][logstash.config.source.multilocal] Ignoring the 'pipelines.yml' file because modu
[2018-07-05T16:09:28,834][INFO ][logstash.runner          ] Starting Logstash {"logstash.version"=>"6.2.4"}
[2018-07-05T16:09:29,786][INFO ][logstash.agent           ] Successfully started Logstash API endpoint {:port=>960
[2018-07-05T16:09:32,941][INFO ][logstash.pipeline        ] Starting pipeline {:pipeline_id=>"main", "pipeline.wor
.delay"=>50}
[2018-07-05T16:09:33,121][INFO ][logstash.pipeline        ] Pipeline started successfully {:pipeline_id=>"main", :
The stdin plugin is now waiting for input:
[2018-07-05T16:09:33,286][INFO ][logstash.agent           ] Pipelines running {:count=>1, :pipelines=>["main"]}
hello udp

仅将文本发送到当前选项卡
ssh://shiyanshi@192.168.1.111:22
E:\tools\java1.8\bin\java ...
收到:{"host":"507","@version":"1","message":"hello udp","@timestamp":"2018-07-05T08:10:06.168Z"}
```

图 6.31 日志经 UDP 方式输出

代码段 6.33：Logstash 配置文件格式（有关 output 的设置）

```
output {
  tcp {
    codec=>…                    #编解码器,可选项,默认为 json
    host=>…                     #字符串,必选项
    mode=>…                      #字符串,取值为"server"、"client"之一,可选项,默认为"client"
    port=>…                     #数值,必选项
    reconnect_interval=>…       #数值,可选项,默认为 10
    workers=>…                  #数值,可选项,默认为 1
  }
}
```

将处理后的部分日志通过 TCP 输出，又分为两种情况，即 output 分别作为客户端和服务器端。

1. 将 Logstash 的 output 作为客户端

首先，设计 Java 应用程序并使之作为服务器端；作为客户端的 Logstash 则基于 TCP 发送处理之后的日志信息到服务器端，TCP 服务器端的代码如代码段 6.34 所示。

代码段 6.34：TCP 服务器端的设计

```
import java.io.*;
import java.net.ServerSocket;
import java.net.Socket;
public class TCPServer_output {
```

```
        public static void main(String[] args) throws IOException {
            ServerSocket serverSocket= new ServerSocket(5656);
            Socket socket= serverSocket.accept();
            InputStream inputStream= socket.getInputStream();
            while (true) {
                byte[] buf= new byte[1024];
                int len= inputStream.read(buf);
                System.out.println(new String(buf, 0, len));
            }
        }
    }
```

其次，设计 Logstash 端的配置文件，如代码段 6.35 所示。注意，这里是设定 Logstash 作为客户端。端口号要和代码段 6.34 中的 ServerSocket serverSocket＝new ServerSocket(5656)设置匹配。

```
代码段 6.35: Logstash 配置文件
input {
  stdin{}
}
output {
  stdout { codec=> rubydebug }
  tcp{
    host=> "localhost"
    port=> 5656
    mode=> "client"
  }
}
```

最后，依次启动 Java 应用程序(服务器端)、Logstash(客户端)。按照代码段 6.34 中的设计思路，服务器端接收在 Logstash 端得到的日志信息。在图 6.32 上部的控制台是 Logstash 端，这里用输入的字符 hello tcp 模拟传输到 Logstash 的日志信息，它们会被传递到服务器端显示(见图 6.32 的下部)。

2. 将 Logstash 的 output 作为服务器端

为验证 Logstash 的 output 作为服务器端的运行效果，这里设计 Java 应用程序并使之作为客户端，如代码段 6.36 所示。

```
[2018-07-05T16:16:21,499][INFO ][logstash.agent          ] Pipelines running {:count=>1, :pipelines=>["main"]}
hello tcp
{
    "@timestamp" => 2018-07-05T08:16:26.354Z,
       "message" => "hello tcp",
          "host" => "507",
      "@version" => "1"
}

仅将文本发送到当前选项卡
ssh://shiyanshi@192.168.1.111:22
E:\tools\java1.8\bin\java "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA 2017.3.4\lib\idea
{"@timestamp":"2018-07-05T08:16:26.354Z","message":"hello tcp","host":"507","@version":"1"}
```

图 6.32　日志经 TCP 方式输出(Logstash 作为客户端)

代码段 6.36: TCP 客户端的设计

```
import java.io.* ;
import java.net.InetAddress;
import java.net.Socket;
public class TCPClient_output {
    public static void main(String[] args) {
        DataOutputStream dos= null;
        BufferedReader brNet= null;
        BufferedReader brKey= null;
        Socket s= null;
        try {
            //建立 Socket
            s= new Socket(InetAddress.getByName("localhost"), 5656);
            InputStream ips= s.getInputStream();
            while (true) {
                byte[] buf= new byte[1024];
                int len= ips.read(buf);
                System.out.println(new String(buf, 0, len));
        }
        //后略
    }
}
```

Logstash 端 output 部分的配置如代码段 6.37 所示。注意,这里设定 Logstash 作为服务器端,端口号要和代码段 6.36 中的 Socket(InetAddress. getByName("localhost"), 5656)匹配。

代码段 6.37：Logstash 配置文件的 output 部分

```
input {
  stdin{}
}
output {
  stdout { codec=>rubydebug }
  tcp{
    host=>"localhost"
    port=>5656
    mode=>"server"
  }
}
```

依次启动 Logstash（服务器端）、Java 应用程序（客户端）。在控制台输入一些字符，用它们来模拟得到的日志信息，它们会经由 Logstash 这个管道以 TCP 方式输送并显示，如图 6.33 所示。在图 6.33 上部的控制台 Logstash 端输入的字符 hello 会传递到客户端显示，见图 6.33 的下部。

```
[2018-07-05T16:24:37,889][INFO ][logstash.agent           ] Pipelines running {:count=>1, :pipelines=>["mai
hello
{
          "host" => "507",
       "message" => "hello",
      "@version" => "1",
    "@timestamp" => 2018-07-05T08:24:58.738Z
}
```

仅将文本发送到当前选项卡

ssh://shiyanshi@192.168.1.111:22

```
E:\tools\java1.8\bin\java ...
{"host":"507","message":"hello","@version":"1","@timestamp":"2018-07-05T08:24:58.738Z"}
```

图 6.33　日志经 TCP 方式输出（Logstash 作为服务器端）

6.5.7　将日志信息发送至 Email

使用 STMP 发送邮件，将收集到的日志信息通过邮件服务器发送至收信人邮箱中，这样可构成一个简单的邮件报警服务。代码段 6.38 给出了 Logstash 配置文件关于邮件发送的部分参数设定。

代码段 6.38：Logstash 配置文件格式，用 STMP 方式发送邮件

```
output {
  http {
```

```
        codec=>…                    #编解码器,可选项,默认为 plain
        address=>…                  #字符串,可选项,默认为 localhost,连接邮件服务器的地址
        body=>…                     #字符串,可选项,发送邮件的文本内容,默认为空字符串
        domain=>…                   #字符串,可选项,邮件服务器的域名,默认为空字符串
        from=>…                     #字符串,可选项,发信人的邮箱,默认为空字符串
        htmlbody=>…                 #字符串,可选项,发送邮件的 html,默认为空字符串
        password=>…                 #字符串,可选项,发信人的密码,默认为空字符串
        port=>…                     #数值,可选项,邮件服务器的端口号,默认为 25
        subject=>…                  #字符串,可选项,发送邮件的主题,默认为空字符串
        to=>…                       #字符串,必选项,收信人的邮箱
    }
}
```

Tips SMTP 是一种邮件传输协议,全称为 Simple Mail Transfer Protocol,即简单邮件传输协议,它通过指定的服务器地址和身份验证将 Email 发送到收信人的邮箱中。使用邮件服务器的前提是需要先开启该服务。例如,本书使用的是腾讯邮件服务器,需要在邮箱的设置中开启该服务。

在代码段 6.39 中,首先接收从命令行输入的内容,经 filter 解析 message,再通过 output 输出到命令行并发送到邮箱中。需要注意的是,在 Email 的配置中,其 subject 和 body 参数都采用"%{变量名}"的格式,将上一步 filter 解析的 message 内容填充到该字符串中。

代码段 6.39:Logstash 配置文件格式,用 SMTP 方式发送邮件

```
input {
    stdin { }
}
filter {
    json {
        source=>"message"
    }
}
output {
    stdout{}
    email {
        to=>'technical@qq.com'              #收信人的邮箱
```

```
            from=> 'monitor@qq.com'                 #发信人的邮箱
            username=> "technical@qq.com"           #发信人的用户名
            subject=> 'Alert-%{title}'              #邮件主题
            body=> "Content:\\n%{content}"          #邮件内容
            domain=> 'smtp.qq.com'                  #邮件服务器域名
            password=> "123456"                     #发信人的密码
            address=> "smtp.qq.com"                 #邮件服务器地址（可填写域名或 IP 地址）
            port=> 25                               #邮件服务器端口
        }
    }
```

如图 6.34 的上部所示，在启动 Logstash 后，命令行输入{"title":"INFO", "content": "hello email"}，filter 解析 message，然后输出到命令行中并发送邮件。在图 6.34 的下部中可以看到，subject 参数的值“Alert - %{title}”已经被替换为“Alert - INFO”，body 中的内容也被替换了。

```
The stdin plugin is now waiting for input:
[2018-07-05T17:26:10,699][INFO ][logstash.agent          ] Pipelines runni
{"title":"INFO", "content": "hello email"}
{
       "content" => "hello email",
      "@version" => "1",
       "message" => "{\"title\":\"INFO\", \"content\": \"hello email\"}",
          "host" => "507",
    "@timestamp" => 2018-07-05T09:26:41.136Z,
         "title" => "INFO"
}
```

Alert - INFO ☆
发件人：[redacted]@qq.com>
时　间：2018年7月5日(星期四) 下午5:26
收件人：[redacted]@qq.com>

Content:
hello email

图 6.34　命令行输入的发送内容和邮件发送结果

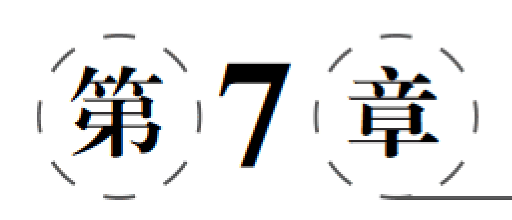
6.6　扩展知识与阅读

redis 是一个高性能的键-值对数据库，它在很大程度上弥补了 memcached 在键-值对存储上的不足，在部分场合可以对关系数据库起到很好的补充作用。有关 redis 的背景知识及操作可参阅文献（李子骅，2013）和（黄健宏，2014）。通过使用 log4j，可以控制日志信息输送的目的地，也可以控制每一条日志的输出格式、定义每一条日志信息的级别等。在得到日志

流以后，可以进行更进一步的分析。文献（王继民，2014）给出互联网用户查询日志挖掘及其应用研究领域的主要技术、方法与实证研究成果。文献（李志义，2015）采用了众多流行的数据挖掘算法，如利用 K-Means 算法进行信息聚类和网页自动抽取，利用贝叶斯分类器实现信息过滤与分类，将智能 Web 算法与网站优化有机地结合起来。

6.7 本章小结

Logstash 架构是专为收集、分析和存储日志而设计的。它的组件架构支持通过代理对不同服务器的日志流进行管理，并最终传送至存储系统中。本章对 Logstash 的主要功能——日志输入输出、过滤处理等进行了说明，并给出了测试与应用实例。由于在 Logstash 中所有的工具都是可安装、可配置以及可管理的，因此，也便于和其他应用对接。将 Elasticsearch、Logstash、Kibana、Beats 等结合起来使用，能有效应对大数据搜索与挖掘方面的应用需求，具有广阔的应用与开发前景。



基于 Kibana 的数据分析及可视化

> Kibana is an open source analytics and visualization platform designed to work with Elasticsearch. You use kibana to search, view, and interact with data stored in Elasticsearch indices. You can easily perform advanced data analysis and visualize your data in a variety of charts, tables, and maps. Kibana makes it easy to understand large volumes of data. Its simple, browser-based interface enables you to quickly create and share dynamic dashboards that display changes to Elasticsearch queries in real time. Setting up Kibana is a snap. You can install Kibana and start exploring your Elasticsearch indices in minutes—no code, no additional infrastructure required.
>
> https://www.elastic.co/guide/en/kibana/current/introduction.html

在对大数据进行分布式索引、检索并对用户日志进行存储和处理后，需要使用信息分析与可视化工具对挖掘结果进行展示。Kibana 是 Elastic Stack 家族中使用 Elasticsearch、Logstash 分析结果的基于 Web 的可视化工具，可以帮助用户汇总、分析和搜索重要数据日志并提供友好的图形界面。例如，通过 histogram 面板，配合不同条件的多个查询，可以对一个事件给出从多个不同维度看的组合统计结果，并可以给出不同的时间序列走势；通过 Kibana 的交互式界面，可以很快定位到异常事件或将其查询范围缩小。Kibana 是用 HTML 和 JavaScript 构建的，所以它不需要任何后台组件。对于 Elasticsearch 用户而言，Kibana 极易上手。Kibana 支持强大的 Lucene query String 查询语法，能利用 Elasticsearch 的过滤、统计功能。本章介绍 Kibana 6.2 在 Logstash 日志数据可视化方面的实际应用。

7.1 Kibana概述

Kibana是一款为协同Elasticsearch工作而设计的开源数据分析和可视化展示平台，用户可以通过Kibana与Elasticsearch索引文件中的数据进行交互，并执行数据检索、数据浏览等任务。在Kibana中可以轻松地完成高级数据分析，以及生成统计图、表格、地图等多种形式的可视化展示。Kibana能够帮助用户理解大数据。它拥有便捷易用、基于浏览器的交互界面，能够通过快速创建和分享动态仪表板，方便地将Elasticsearch中执行查询的情况展示给用户。

安装Kibana的过程十分简单，在安装Kibana之后，无须编写任何代码，也不需要任何底层支持，即可对Elasticsearch分片中的数据进行操作和管理。

Kibana 6.2包括Discover、Visualize、Dashboard、Timelion[①]、Dev Tools和Management 6个组件。

7.2 安装Kibana

Elastic官网的Kibana产品页面(https://www.elastic.co/downloads/kibana)提供了Kibana针对不同平台的安装包。首先，根据当前系统平台的实际需要，从Elastic官网下载新版的Kibana(本章以Kibana 6.2为例)，然后将其解压。在config文件夹中的kibana.yml配置文件中可以配置和Kibana连接的Elasticsearch信息(默认连接本机的9200端口，而Kibana运行在5601端口；如果不修改配置文件，则使用默认配置)。进入Kibana主目录中的bin文件夹，启动Kibana，在浏览器地址栏中输入http://localhost:5601/以访问Kibana的前端界面。图7.1为Kibana的初始前端界面。

> Tips Kibana需要以Elasticsearch中存储的数据作为数据源，原则上应事先确保以下3点：Elasticsearch正在运行；在Elasticsearch中已经创建了索引文件；在索引文件中已经存有数据。如果Kibana启动时并没有检测到任何可用的Elasticsearch连接，那么即使能够启动Kibana并访问前端页面，其状态也将变为Red，如图7.2所示，而数据检索和分析等进一步的处理工作通常也无法执行。

① 时间线的英文应为timeline。为便于读者阅读，本书仍使用Kibana中的写法——timelion。

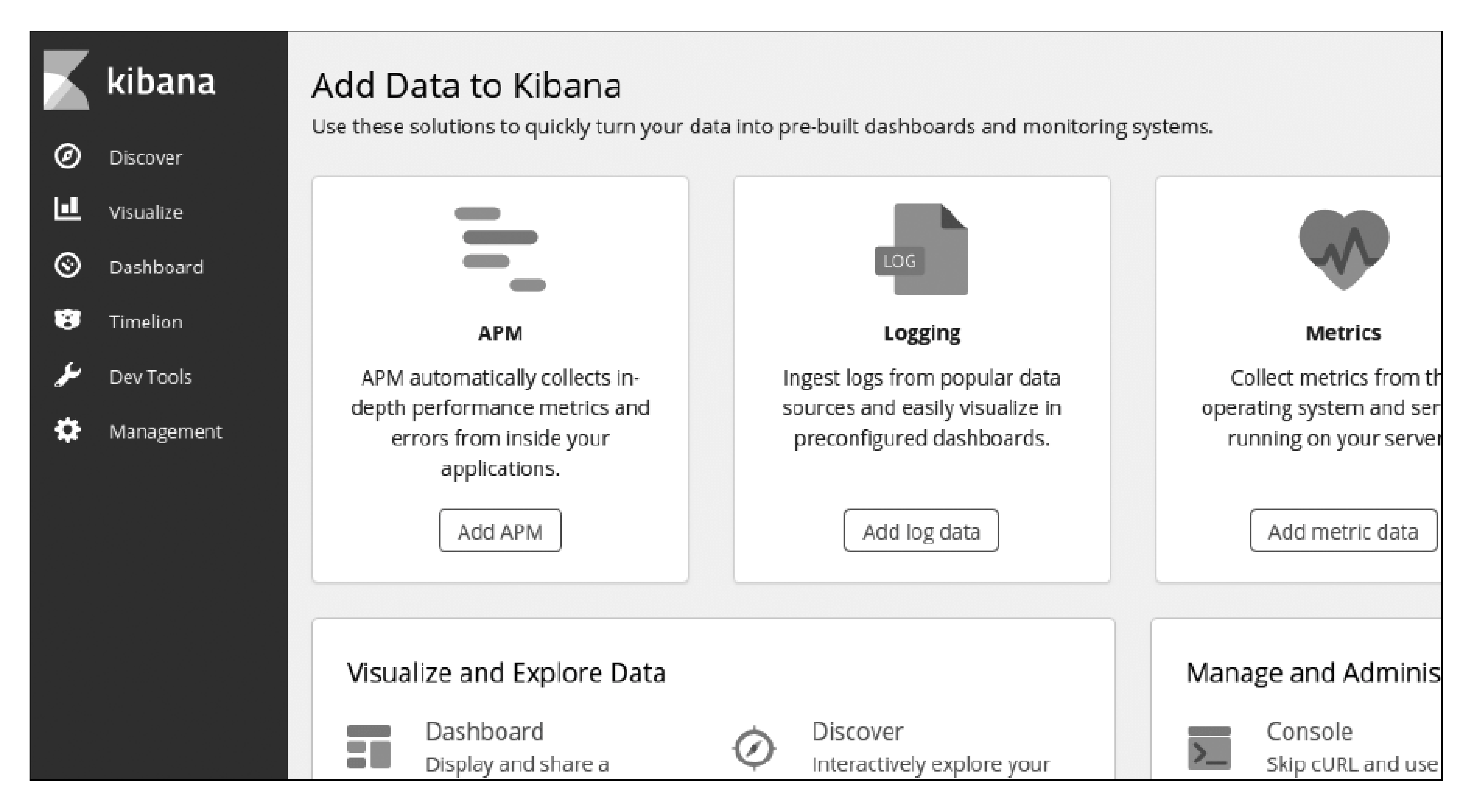


图 7.1　Kibana 的初始前端界面

Status: Red

Heap Total
112.40 MB

Heap Used
77.04 MB

Response Time Avg
0.00 ms

Response Time Max
0.00 ms

Status Breakdown

ID	Status
plugin:kibana@6.2.4	✔ Ready
plugin:elasticsearch@6.2.4	⚠ Unable to connect to Elasticsearch at http://localhost:9200.
plugin:timelion@6.2.4	✔ Ready
plugin:console@6.2.4	✔ Ready
plugin:metrics@6.2.4	✔ Ready

图 7.2　当至少缺失一个主节点时，Kibana 状态将变为 Red

7.3　使用 Management 组件管理配置

Management 组件可以用来管理 Kibana 的运行时配置，包括对索引的引用和进一步设置，调整 Kibana 自身运行状态的高级设置，以及配置 Kibana 中存储的检索、可视化内容、仪表

板等对象。在 Kibana 前端界面左侧单击 Management 导航按钮即可跳转到 Management 界面，如图 7.3 所示。本节对添加索引模式、高级设置、已保存对象管理等功能进行介绍。

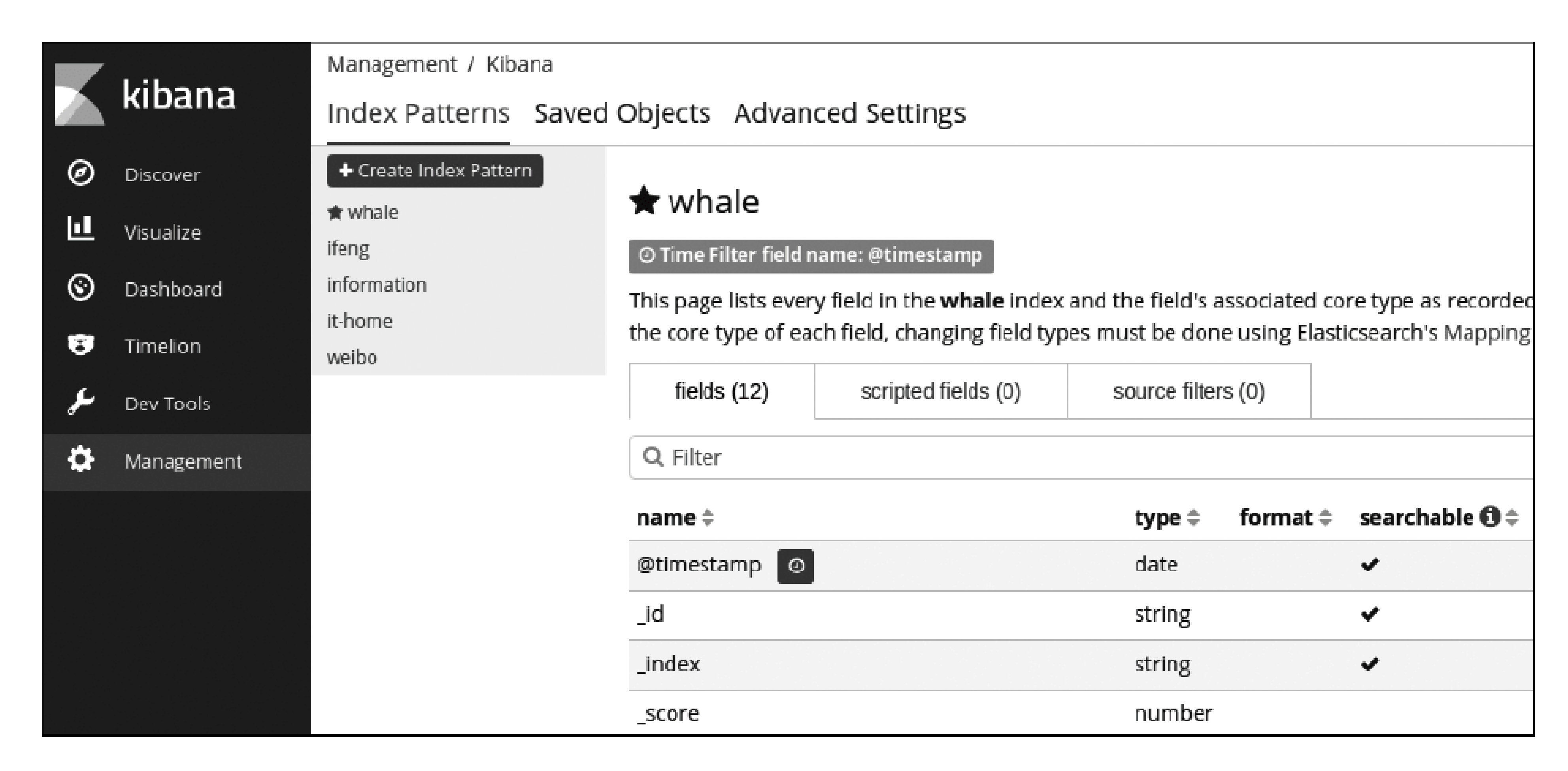


图 7.3 Management 界面

7.3.1 创建索引模式

Kibana 的数据分析和展示能力要通过来自 Elasticsearch 的数据源才能得以发挥。首先要向 Kibana 中引入索引模式。在 Index Patterns 界面中输入要添加的索引名称，下方会列出目前 Elasticsearch 已有的索引。在用户输入某个索引名称的过程中，程序会根据当前已输入的字符串自动筛选可用的索引名称。Management 允许用户使用通配符来指定索引名称，即在输入的内容中可以加入通配符 *（例如，logstash-* 表示名称以字符串 logstash-开头的全部索引）。当用户准确输入了索引的名称之后，界面中会出现匹配成功的提示，如图 7.4 所示。此时单击 Next step 按钮即可进行第二步操作。

第二步是选择时间字段。当索引中存在时间类型的数据时，Kibana 可以执行与时间相关的查询、统计和聚合。界面中提供了一个选择时间字段的下拉列表，其中列出了当前指定的索引中的全部时间字段。用户可以选择一个起决定作用的时间字段，也可以选择最后的一项，即不使用时间过滤器(此时用户不需指定时间字段)。当用户在下拉列表中选定一项之后，右侧的 Create index pattern 按钮即变为可用状态，如图 7.5 所示，单击该按钮即可完成索引模式的创建。

创建索引模式后，界面中将出现该索引的基本信息。在索引名称右侧有 3 个操作按钮，它们的作用如下：单击左侧的 ★ 按钮，可将该索引设置为默认；单击中间的 ↻ 按钮，可刷新下面的字段列表；单击右侧的 🗑 按钮，可将该索引从界面中移除。

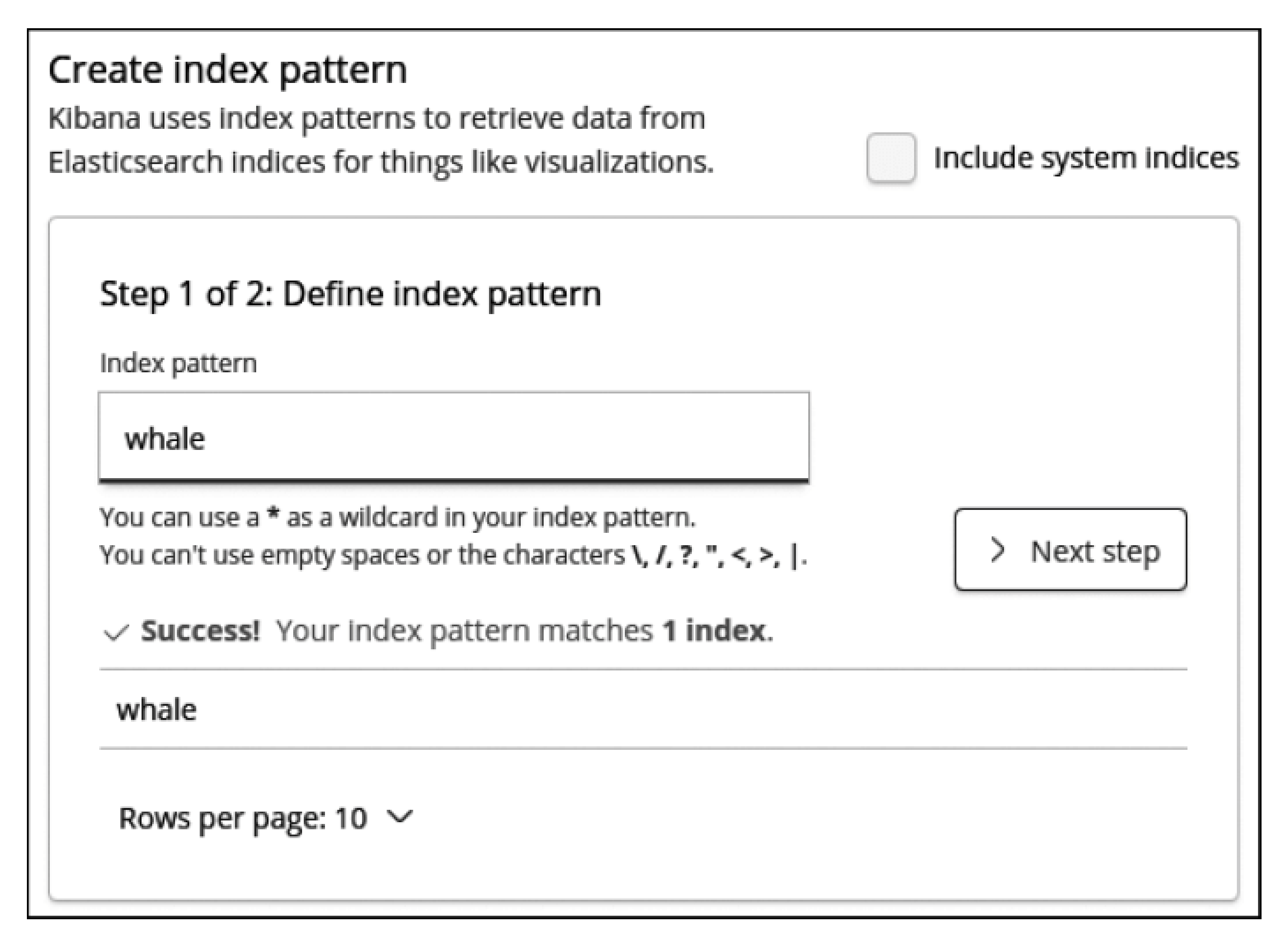


图 7.4　创建索引模式第一步：指定索引

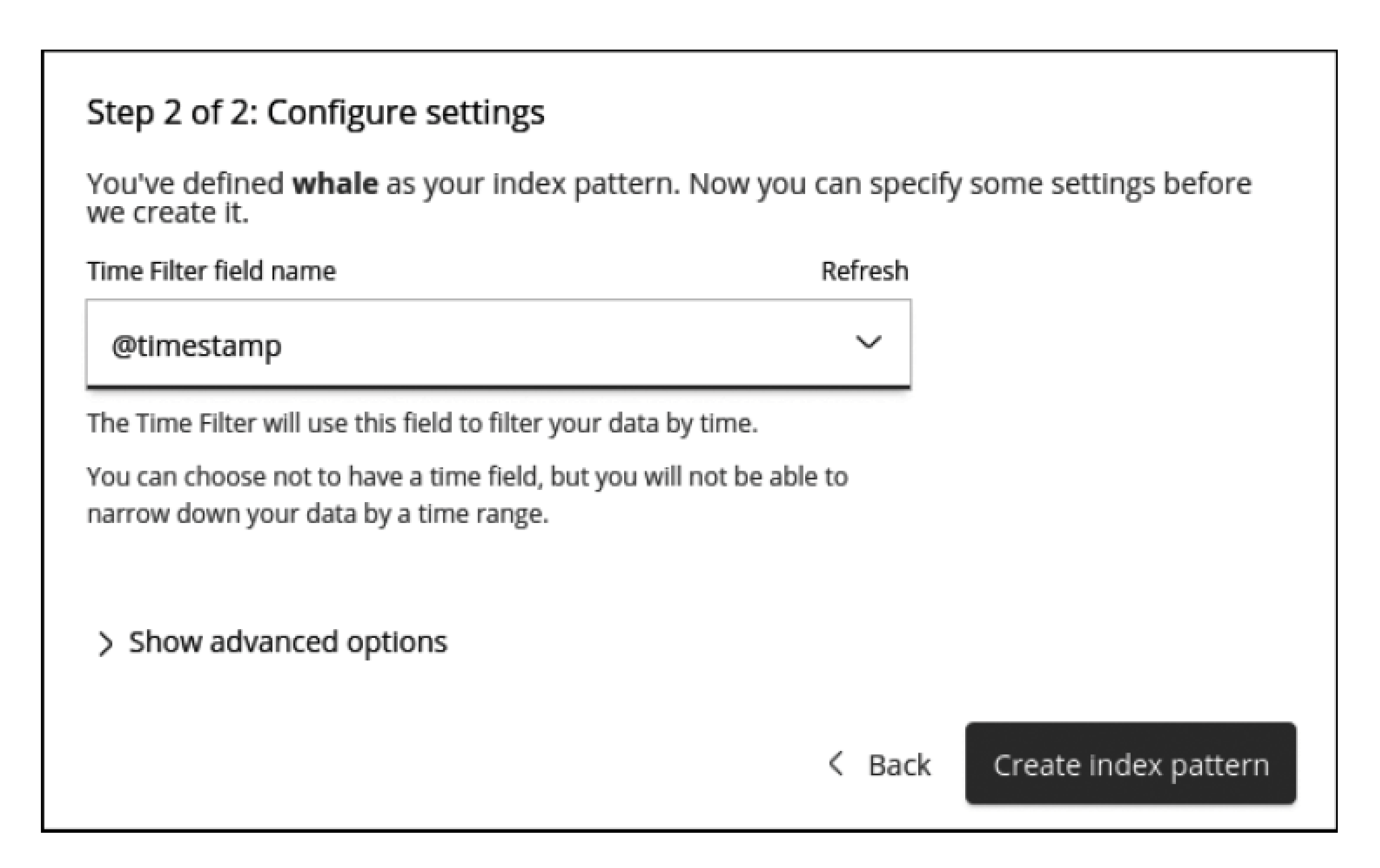


图 7.5　创建索引模式第二步：选择时间字段

创建至少一个索引模式之后，可以看到界面左侧为索引模式的列表，右侧为选中的索引模式的基本信息。单击界面左上方的 + Create Index Pattern 按钮，可以创建更多的索引模式，具体操作方法同上，不再赘述。

7.3.2　高级设置

Advanced Settings（高级设置）页面提供了直接控制 Kibana 运行状态的设置项，通过直

接编辑这些设置项,可以实现更改日期格式、指定默认索引模式、设置小数数值精度等功能。在 Kibana 前端界面中单击 Advanced Settings 导航按钮即可跳转至高级设置界面。下面对高级设置中的设置项进行简要介绍,详细内容可参考相关文献和互联网上的资源。

- query:queryString:options:关于 Lucene 中 queryString 分析器的设置项,默认为{ "analyze_wildcard": true }。
- sort:options:关于 Elasticsearch 中排序参数的设置项,默认为{ "unmapped_type": "boolean" }。
- dateFormat:用于显示统一格式的日期时间,默认为“MM DD YYYY, HH:mm:ss.SSS”。
- dateFormat:tz:Kibana 中使用的时区,默认为 browser(即沿用浏览器的时区设置)。
- dateFormat:scaled:设置时间戳数据格式,格式化的时间信息必须在 ISO 8601 标准规定的范围内,默认为[["", "HH:mm:ss.SSS"], ["PT1S", "HH:mm:ss"], ["PT1M", "HH:mm"], ["PT1H", "YYYY-MM-DD HH:mm"], ["P1DT", "YYYY-MM-DD"], ["P1YT", "YYYY"]]。
- dateFormat:dow:规定一周的第一天,默认为 Sunday(星期日)。
- defaultIndex:指定默认的索引,默认为 null,该项可删除。
- defaultColumns:指定在 Discover 页面中默认显示索引中的哪些列(即字段),默认为_source,即全部自定义字段。
- metaFields:以数组形式指定索引中_source 字段以外的其他字段,即 Elasticsearch 自动生成的元字段。Kibana 在显示文档数据时,将这些字段与_source 字段合并进行显示,默认为_source, _id, _type, _index, _score。
- discover:sampleSize:指定在 Discover 界面中显示数据的行数,默认为 500。
- doc_table:highlight:指定是否允许在 Discover 界面及已保存的检索面板中高亮显示搜索结果,默认为 true。
- courier:maxSegmentCount:Kibana 发送请求到 Elasticsearch 时会将请求分段发送,以减小每段请求信息的长度。该项指定每个分段的长度,默认为 30。
- fields:popularLimit:指定显示最热字段的最大数量,默认为 10。
- histogram:barTarget:指定当 date histogram 使用了自动时间间隔时,Kibana 尝试生成条形统计图中条带的数量,默认为 50。
- histogram:maxBars:指定上一项的最大值,默认为 100。
- visualization:tileMap:maxPrecision:指定在 tile map 中显示的最大 geoHash 精度。7 表示高,10 表示很高,12 为最大值,默认为 7。

- visualization:tileMap:WMSdefaults：关于支持 tile map 的 wms map server 的默认属性，默认为{ "enabled": false,"url": "https://basemap.nationalmap.gov/arcgis/services/USGSTopo/MapServer/WMSServer","options": { "version": "1.3.0","layers": "0","format": "image/png","transparent": true,"attribution": "Maps provided by USGS","styles": "" } }。
- visualization:colorMapping：将可视化界面中的某个值映射到指定的颜色，默认为{"Count":"#6eadc1"}。
- visualization:loadingDelay：指定在查询时将可视化界面变暗之前等待的时间，默认为 2s。
- csv:separator：指定导出数据的分隔符，默认为逗号。
- csv:quoteValues：指定是否引用导出的信息，默认为 true。
- history:limit：在具有历史数据的字段中，指定显示历史数据的数量，默认为 10。
- shortDots:enable：指定可视化界面中是否将较长的字段名以简短的形式显示，例如将 foo.bar.baz 简化为 f.b.baz，默认为 false。
- truncate:maxHeight：指定表格中单元格的最大高度，设置为 0 以禁用截断功能，默认为 115。
- indexPattern:fieldMapping:lookBack：指定最近匹配到索引模式的数量，来查询包含时间戳的索引模式中字段的 mapping，默认为 5。
- format:defaultTypeMap：指定默认的字段类型映射，如果某个字段未匹配到指定的映射，那么该字段使用默认映射，默认为{ "ip": { "id": "ip","params": {} },"date": { "id": "date","params": {} },"number": { "id": "number","params": {} },"boolean": { "id": "boolean","params": {} },"_source": { "id": "_source","params": {} },"_default_": { "id": "string","params": {} } }。
- format:number:defaultPattern：指定 number(数字)格式默认的数值型格式，默认为 0,0.[000]。
- format:bytes:defaultPattern：指定 bytes(字节)格式默认的数值型格式，默认为 0,0.[000]b。
- format:percent:defaultPattern：指定 percent(百分数)格式默认的数值型格式，默认为 0,0.[000]%。
- format:currency:defaultPattern：指定 currency(货币)格式默认的数值型格式，默认为($0,0.[00])。
- savedObjects:perPage：指定已保存的 object 列表中每页的 object 数量，默认为 5。
- timepicker:timeDefaults：Kibana 中默认的时间过滤器配置，默认为{ "from":

"now-15m","to": "now","mode": "quick" }。

- timepicker:refreshIntervalDefaults：时间过滤器默认的刷新间隔，默认为{ "display": "Off","pause": false,"value": 0 }。
- dashboard:defaultDarkTheme：指定新创建的面板是否使用暗色调主题，默认为false。
- filters:pinnedByDefault：指定过滤器是否应被固定为具有全局状态，默认为 false。
- notifications:banner：以横幅形式面向全体用户定制临时公告，支持 Markdown 格式。
- notifications:lifetime:banner：指定临时公告的显示时长，单位为毫秒，设置为 Infinity(即无穷大)将禁用公告横幅，默认为 3 000 000。
- notifications:lifetime:error：指定错误报告的显示时长，单位为毫秒，设置为 Infinity 将禁用公告横幅，默认为 300 000。
- notifications:lifetime:warning：指定警告的显示时长，单位为毫秒，设置为 Infinity 将禁用公告横幅，默认为 10 000。
- notifications:lifetime:info：指定通知信息的显示时长，单位为毫秒，设置为 Infinity 将禁用公告横幅，默认为 5000。
- timelion:showTutorial：指定在用户进入 timelion 时是否显示教程，默认为 false。
- timelion:es.timefield：指定在使用.es()方法时默认包含时间戳的字段名，默认为@timestamp。
- timelion:es.default_index：指定在使用.es()方法时默认在 Elasticsearch 中搜索的索引名称，默认为_all。
- timelion:target_buckets：指定在使用自动间隔大小时获取 bucket 的数量，默认为 200。
- timelion:max_buckets：指定单个 datasource 返回 bucket 的最大数量，默认为 2000。
- timelion:default_columns：指定 timelion 表中默认的列数，默认为 2。
- timelion:default_rows：指定 timelion 表中默认的行数，默认为 2。
- timelion:graphite.url：指定 graphite 监控系统主机的 URL 地址，默认为 https://www.hostedgraphite.com/UID/ACCESS_KEY/graphite。
- timelion:quandl.key：指定 quandl 金融和经济数据网站 www.quandl.com 的 API key，默认为 someKeyHere。
- state:storeInSessionStorage：为防止 URL 过长导致某些浏览器无法正常处理，可以将 URL 的一部分存储在服务器 session 中，该设置项用于指定是否开启这一功

能，默认为 false。

- discover:aggs:terms:size：指定 Discover 中，在聚合时词项的最大长度，默认为 20。

> Tips 对以上设置项的修改，将会对 Kibana 的性能造成明显的影响，有可能会造成难以排查的问题。当某个设置项不填写任何内容保存时，其值会恢复默认设置。如果其他设置项需要这里的设置，那么恢复默认将造成配置的不兼容。另外，对于某个设置项的删除操作是不可逆的，操作将永久生效。

7.3.3 管理已保存的检索、可视化和仪表板

在 Management 组件中，单击上方的 Saved Objects 链接，可以跳转至已保存对象的管理界面，如图 7.6 所示。管理内容分为仪表板、检索和可视化内容（分别对应页面中的 Dashboards、Searches、Visualizations）。在该界面中每一种内容可以显示 100 条信息，其余部分可以在界面上方的输入框中通过输入部分对象名称过滤得到。

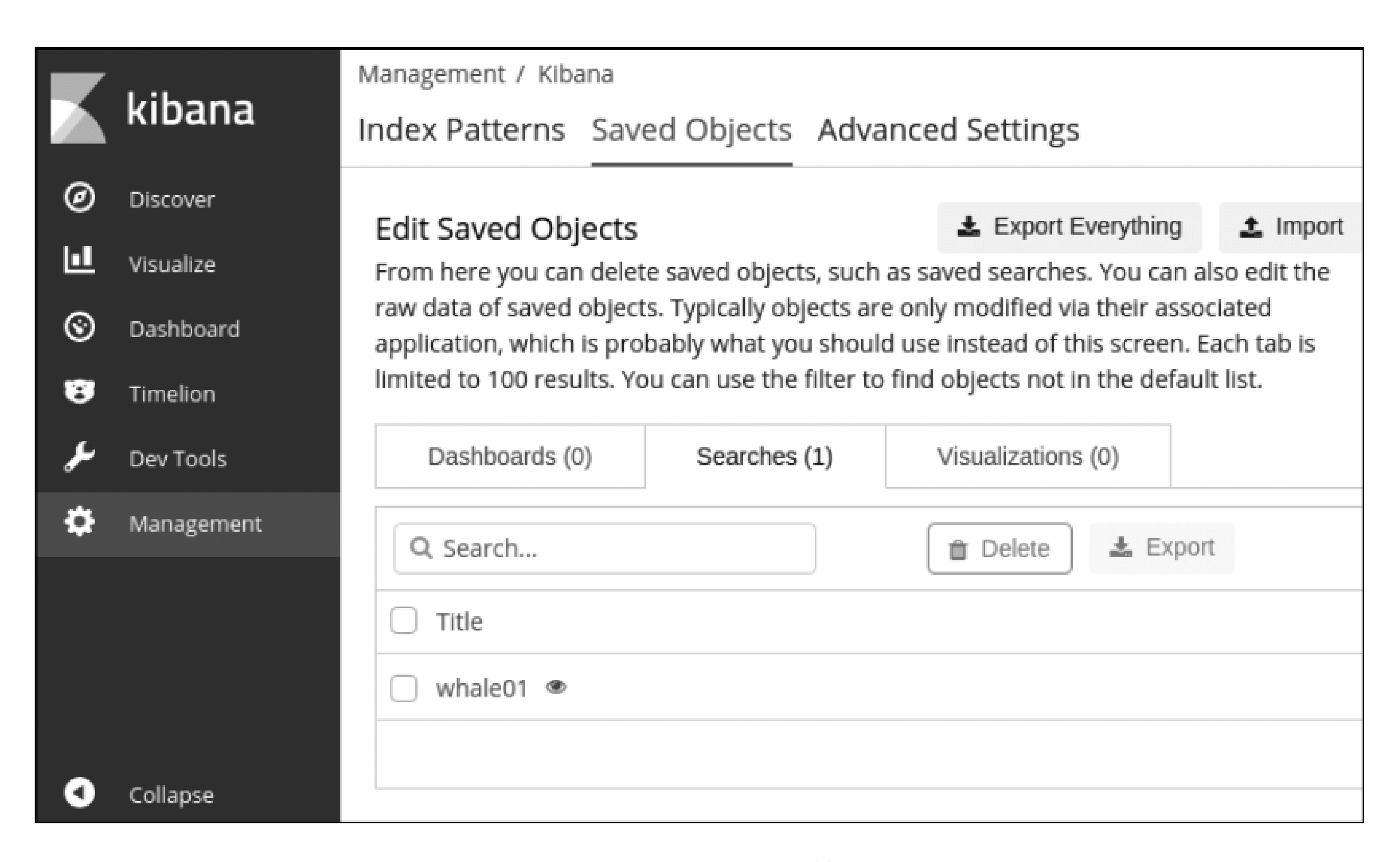


图 7.6 Saved Objects 管理界面

在每种对象的列表中，左侧为保存的对象名称，鼠标指针放在该行时其右侧会出现查看对象内容的按钮。单击对象名称可以跳转至该对象的编辑界面，单击右侧的按钮可以跳转至该对象的内容查看页面。单击列表左上方的勾选项，可以将保存的对象全部选中；单击列表顶部的 Delete 按钮，可以将选中的对象删除；单击 Export 按钮，可以将选中的对象导出并以 JSON 格式的文件保存到本地。

Management 组件在屏幕的最上方还提供了输出所有对象的 Export Everything 按钮和导入外部保存对象的 Import 按钮。该管理界面功能十分简单易用，本节不再赘述。

7.4　使用 Discover 组件执行查询

在 Kibana 前端界面左侧单击 Discover 导航按钮即可跳转到 Discover 界面，如图 7.7 所示。本节将对时间过滤器的设置、数据查询、字段过滤、查看文档数据、查看统计信息等功能进行介绍。Discover 提供了交互式的查询界面，可以在已创建的索引模式中查询索引文件中的全部文档信息。在界面中还提供了执行查询语句、执行查询结果过滤、查看文档全部数据、显示相关统计数据等功能。当索引模式中带有时间戳字段时，查询界面上方会显示柱状统计图。



图 7.7　Discover 界面

7.4.1　设置时间选择器

对于带有时间戳的文档而言，时间选择器能够在某个时间段内统计文档中的数据。进入 Discover 中的查询页面时，默认加载最近 15min 的统计数据。如要更改时间段，可以单击界面右上角的时间选择器按钮，界面上方会弹出时间选择界面。选择时间段的方式主要有 3 种：在给定的常用时间段中快速选择，设置相对于当前的起始时间，从起始和终止日历中设置绝对时间间隔。

单击左侧的 Quick 按钮可以转到快速选择界面，该界面中提供了 22 种常用时间段，包括截至目前的各种时间跨度、类似上周或上个月的特定时间跨度等。该界面中的选项可以

快速度划定时间段并完成统计。在选择时间跨度之后，还可以在 Discover 查询界面中的柱状图中通过鼠标左键拖曳的方式来圈出更精确的时间范围。如图 7.8 所示，图中矩形的阴影区域即为鼠标拖出的时间范围。此外，将鼠标指针放在柱状图中任意一个小格上，均会显示出该位置详细的时间和统计信息。如果单击该位置，统计图会展开为这一单位时间跨度内更详细的数据的统计图。

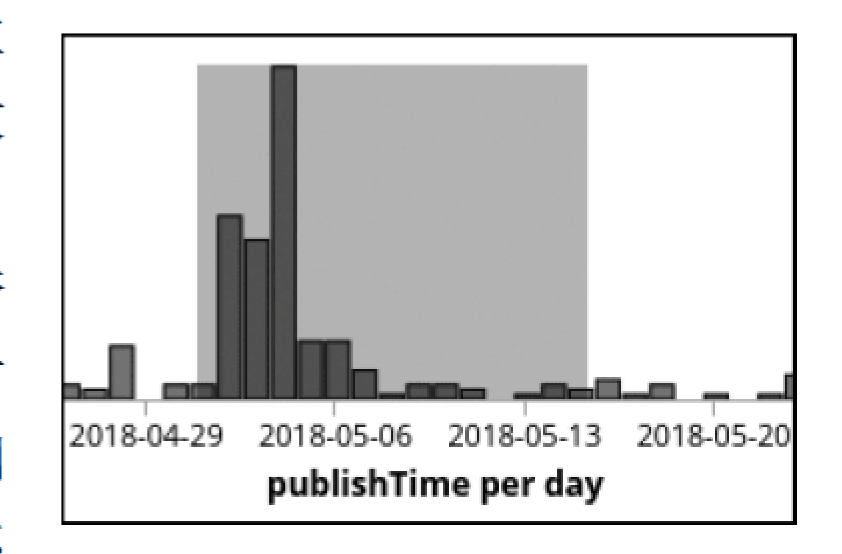
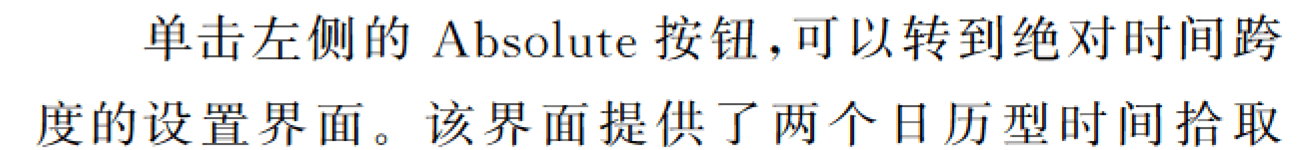


图 7.8　在柱状图上直接拖曳来划定精确的时间范围

单击左侧的 Relative 按钮，可以转到相对时间跨度的设置界面。在该界面中可以设置历史上的一个时间点，从而划定距今的时间跨度。界面中给出了小到分、大到年的 7 种不同的时间单位，如有特殊需要，还可以勾选四舍五入到已选择时间单位的选项。



单击左侧的 Absolute 按钮，可以转到绝对时间跨度的设置界面。该界面提供了两个日历型时间拾取器，用来为两个时间输入框选择准确的起止时间，从而组成一个特定的时间跨度。另外，每个时间输入框的下方均给出了要求的 SimpleDateFormat 时间格式。在终止时间输入框上方还提供了设置当前时间的 Set To Now 按钮，单击该按钮可以将终止时间设置为当前时间。



单击浏览器的后退按钮，可以撤销 Kibana 中的上一步操作。

7.4.2　在索引模式中执行搜索

Discover 界面上方为用户提供了一个搜索输入框，其右侧带有一个执行搜索的按钮，用来在当前显示的索引文件中执行各类搜索任务。搜索框中填入的表达式应符合 Elasticsearch 中 query string 查询的格式。表 7.1 给出了该格式查询表达式的一些实际例子。

表 7.1　query string 查询表达式实例

输入表达式	表达式含义
os:Linux	检索字段 os 中带有关键词 Linux 的文档内容
os:"Linux"	检索字段 os 中带有精确词项 Linux 的文档内容
os:(Linux OR Windows 10)	检索字段 os 中带有关键词 Linux 或关键词 Windows 10 的文档内容

续表

输入表达式	表达式含义
Win?ows *	通配符?代表一个任意字符，* 代表零到任意多个任意字符。该表达式检索以 Win 开头，中间带有一个任意字符，后面接 ows 和任意后续内容的文档内容
log_size:[300 TO 500]	检索字段 log_size 中文档长度为 300～500 个字符的文档内容
log_size:[400 TO *]	检索字段 log_size 中文档长度在 400 以上的文档内容
log_size:[300 TO 600] AND os:"Linux"	检索字段 log_size 中文档长度为 400～600 个字符并且字段 os 中带有精确词项 Linux 的文档内容

Kibana 提供了保存查询结果的功能。在查询出结果之后，单击界面上方的 Save 按钮，为当前要保存的查询起一个名字，即可保存该结果。如果要将已保存的查询打开，也可以单击界面上方的 Open 按钮，将会列出之前保存的所有查询，单击查询的名称即可显示出查询结果。要执行一个新的查询，可以单击界面上方的 New 按钮。

随着越来越多的文档添加到当前的索引模式中，Discover 界面中的查询结果可能需要更新。如果用户需要实时更新查询结果，可以单击界面右上角的时间选择器按钮，界面上方展开时间设置面板后，在时间选择器按钮左侧会出现一个自动刷新按钮 Auto-refresh 。单击这一按钮，时间设置面板中会给出从 5s 到 1d 的 12 种刷新间隔时间长度以及一个 Off(关闭)按钮。单击其中一个时间长度后，将以此处设定的时间间隔定时刷新，同时右上方的自动刷新按钮变为暂停按钮。如果要停止刷新，可以单击暂停按钮，或者直接将自动刷新设置为关闭状态。

7.4.3　字段过滤

加载过索引模式后，Discover 界面左侧提供了当前索引文件中的字段列表，由 Selected Fields(已选字段)和 Available Fields(可用字段)两部分组成，并且可用字段中会将经常被选中的字段指定为 Popular(常用字段)，如图 7.9 所示。在任意一个字段名上单击，其下方将会展开该字段统计数量最高的前 5 种内容，如图 7.10 所示。

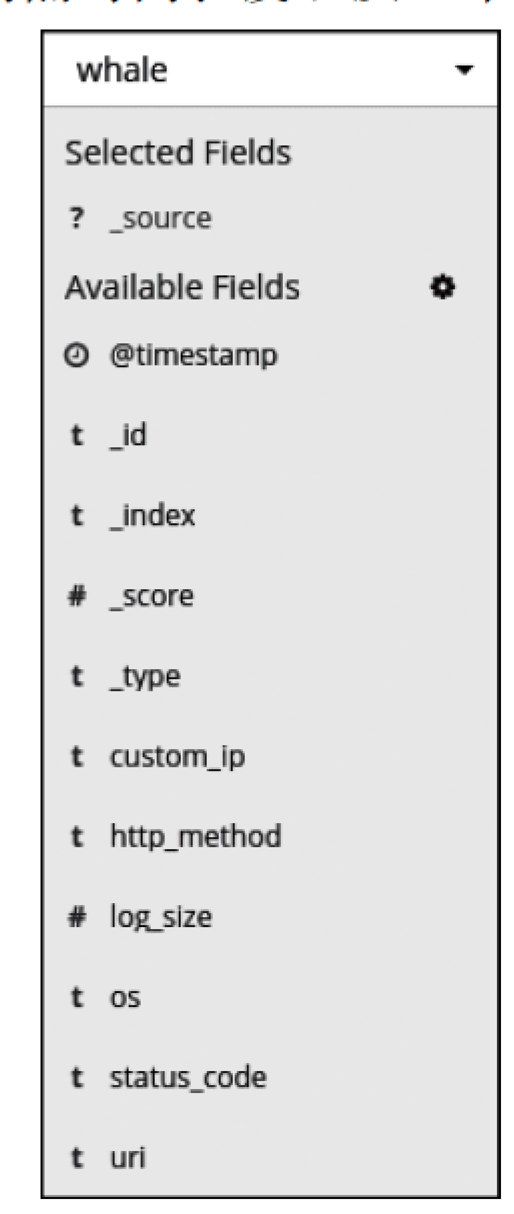


图 7.9　当前索引文件中的字段列表

在每一行统计数据右侧均有两个过滤器按钮和。单击按钮，可以将已选择的字段和内容作为精确的检索条件执行检索，该条件也将以过滤器标签的形式显示在搜索输入框下方；单击按钮，则会将该数据从总体数据中排除。将鼠

标放置在标签上，标签将显示出 5 种操作按钮，分别为启动/禁用按钮、固定按钮、检索/排除转换按钮、移除按钮和编辑按钮，如图 7.11 所示。

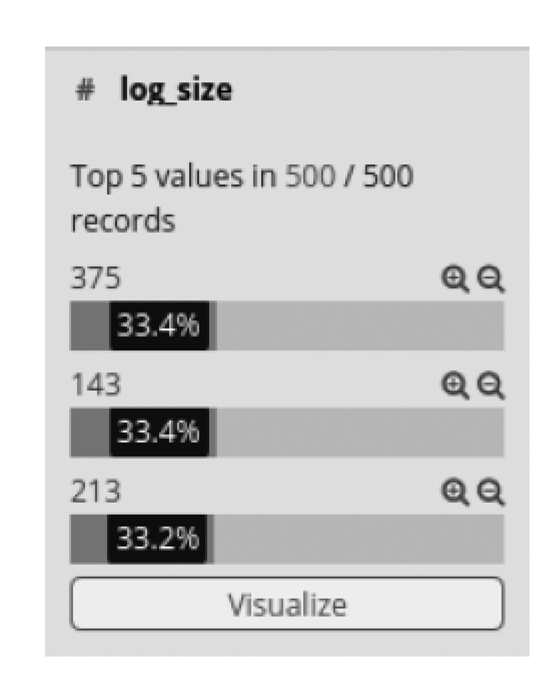


图 7.10　字段内容的统计数据

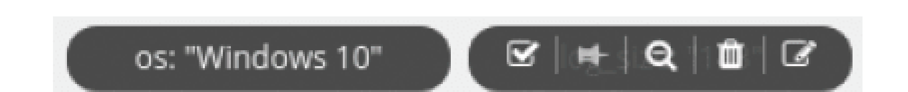


图 7.11　当前索引文件中的字段

单击按钮，可以在不删除该过滤器标签的情况下禁用该过滤器功能，禁用时该标签将显示为斜纹背景，再次单击即可重新启用。单击按钮，可以使该标签在界面发生跳转时仍与检索保持联系。例如，当转到可视化界面时，被固定的标签将继续在新的界面中起作用，但是检索内容不包含过滤器标签指定字段的情况除外。单击按钮，可以实现上面所述的添加条件到检索当中或排除性检索的转换功能。单击按钮，可以将当前过滤器标签移除。单击按钮，可以对当前过滤器的执行代码进行编辑，以及为该标签定义别名，别名将替换图 7.11 左侧标签中的文字。

如果要在界面中部的文档内容列表中添加过滤器，可以单击任意一条文档数据左侧的三角形按钮，可将当前文档内容展开，随后即可如上文所述使用过滤器按钮和进行相应的检索或排除操作。此外，单击右侧的 * 按钮，可以添加一个设置该列是否存在的过滤器标签，标签用法同上。

7.4.4　查看文档数据

在每次查询时，Discover 界面中部均会显示前 500 条文档数据，默认每条数据均显示 _source 字段，即全部自定义添加的字段。用户也可以在 Management 界面中设置 discover:sampleSize 属性来修改默认显示文档的数量。单击任意一条文档开头的按钮，展开文档内容后，即可查看该文档所包含的详细信息（包括 Table 和 JSON 两种格式），如图 7.12 和图 7.13 所示。

如需对文档内容进行排序，可以通过在文档内容列表的表头单击向上或向下的三角形按钮来完成，反复单击该按钮可以在正序和倒序之间切换。

如上文所述，加载了索引模式后的 Discover 界面左侧提供了当前索引文档中的字段列

Table	JSON
@timestamp	January 15th 2018, 07:59:30.000
t _id	pF2kLWQBg1CkS_KXCtXZ
t _index	whale
# _score	-
t _type	_doc
t custom_ip	127.0.0.1
t http_method	GET
# log_size	375
t os	Linux
t status_code	200
t uri	/dispatching/commons/api/callba Fj_vY6eUwsQql&spiderUUID=d62e99 9ad5a2 HTTP/1.1

图 7.12　当前索引文件中的字段(Table 格式)

Table　JSON


```
{
  "_index": "whale",
  "_type": "_doc",
  "_id": "pF2kLWQBg1CkS_KXCtXZ",
  "_version": 1,
  "_score": null,
  "_source": {
    "@timestamp": "2018-01-14T23:59:30",
    "http_method": "GET",
    "status_code": "200",
    "os": "Linux",
    "log_size": 375,
    "custom_ip": "127.0.0.1",
    "uri": "/dispatching/commons/api/callback?c
        -4a4e-4951-b2d8-35d8e29ad5a2&taskId=d62
  },
  "fields": {
    "@timestamp": [
      "2018-01-14T23:59:30.000Z"
    ]
  },
  "sort": [
    1515974370000
  ]
}
```


图 7.13　当前索引文件中的字段(JSON 格式)

表。默认所有字段全部显示在可用字段中。当鼠标指针放置在任意一个字段上面时,列表右侧均会出现一个 add 按钮,如图 7.14 所示。单击该按钮,则当前指定的字段将显示在已选字段中,同时界面中每个文档的详细信息将只保留已选字段的内容。在展开的文档详细信息中,单击◫按钮,可以设置所有文档信息只保留已选字段的信息;再次单击该按钮,即可将所有文档恢复原状。

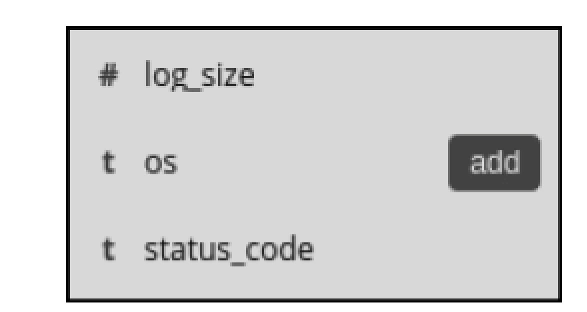


图 7.14　字段列表中的 add 按钮

7.5 使用 Visualize 组件创建统计图表

Visualize 组件提供了从 Elasticsearch 分片中创建数据可视化统计图表的功能。这些可视化内容基于对 Elasticsearch 的查询，同时可以使用一系列聚合，以便从数据中提取和处理有用的信息。所有这些查询的结果都能以统计图表的形式展示在仪表板中，从而便于用户获得数据的变化趋势。要创建数据可视化统计图表，可以通过在 Discover 中已保存的检索或建立新的查询来执行。在 Kibana 前端界面左侧单击 Visualize 导航按钮，即可跳转到 Visualize 界面，如图 7.15 所示。本节对可视化统计图表的创建进行介绍。

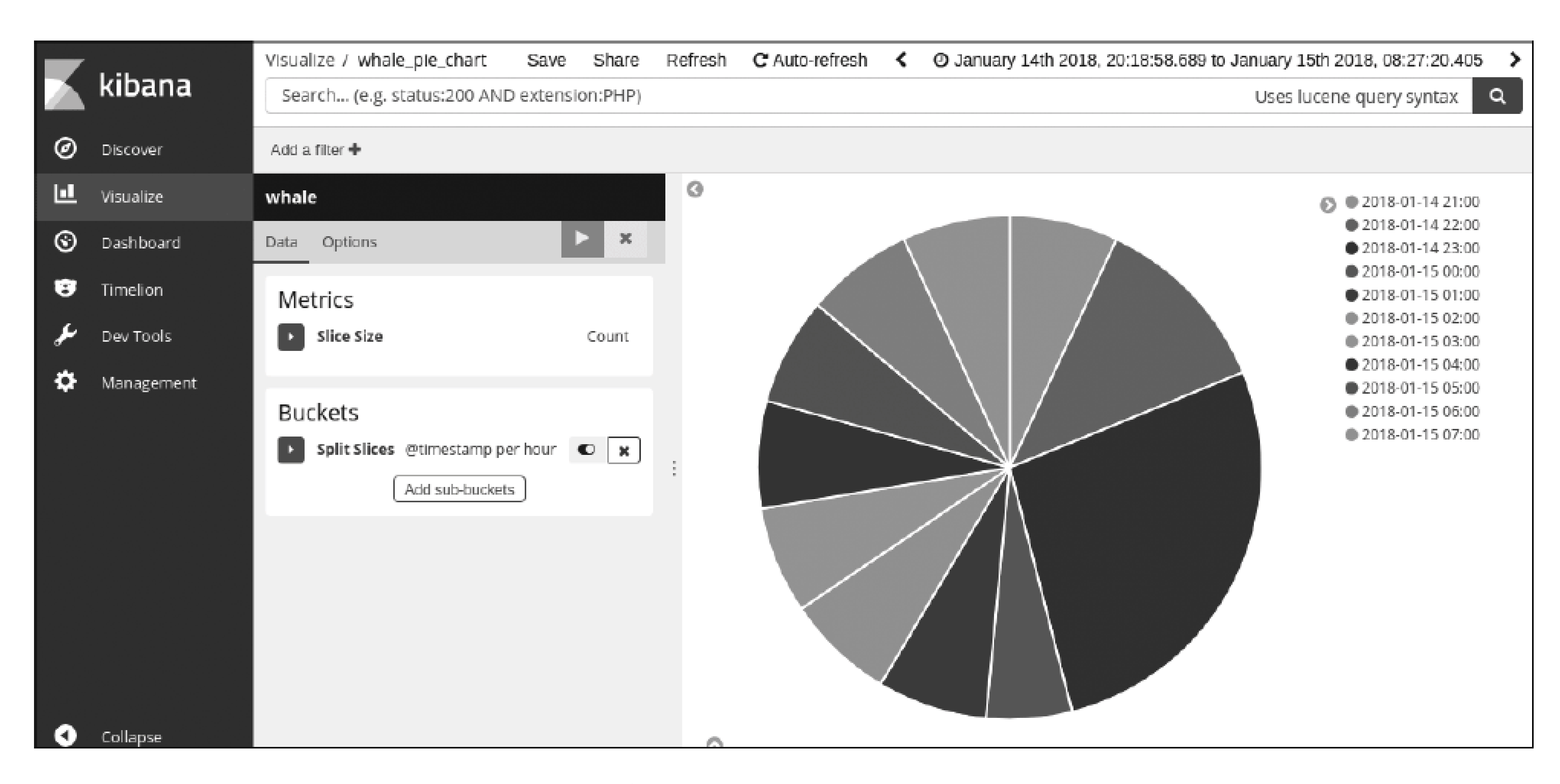


图 7.15　Visualize 界面

单击 Kibana 界面左侧的 Visualize 导航按钮进入 Visualize 后，界面中会出现添加可视化内容的按钮，单击该按钮后，界面中会列出 18 种类型的统计图表，分别为面积图、热力图、横向条形图、折线图、饼图、竖向条形图、数据表、gauge 仪表板、goal 仪表板、统计数值、坐标地图、区域地图、时间线、可视化编辑器、控件、Markdown 文本解析器部件、标签云、Vega。选择其中任意一个类型，即可转到索引模式选择页面。在这里选择需要创建统计图表的数据的索引模式，可以通过选择界面中列出的索引模式名称执行新的查询来获取数据，也可以通过选择之前已保存的检索来直接获取数据，这两种方式分别位于界面的左右两边。接下来只需为统计图表指定各个坐标轴上的数据，即可生成统计图表。各类可视化统计图表的创建方法大致相同。下面以统计数值（Metric）和标签云（Tag cloud）为例，对基本的统计图表生成方法进行介绍。

在 Visualize 界面中单击 Metric 选项，进入索引模式选择界面。在列表中选择一个索

引文件或已保存的检索，即可转到可视化创建界面中。界面左侧是可视化数据的设置部分。在这里设置统计数值的参数后，可视化效果即展示在界面右侧区域中，如图 7.16 所示。

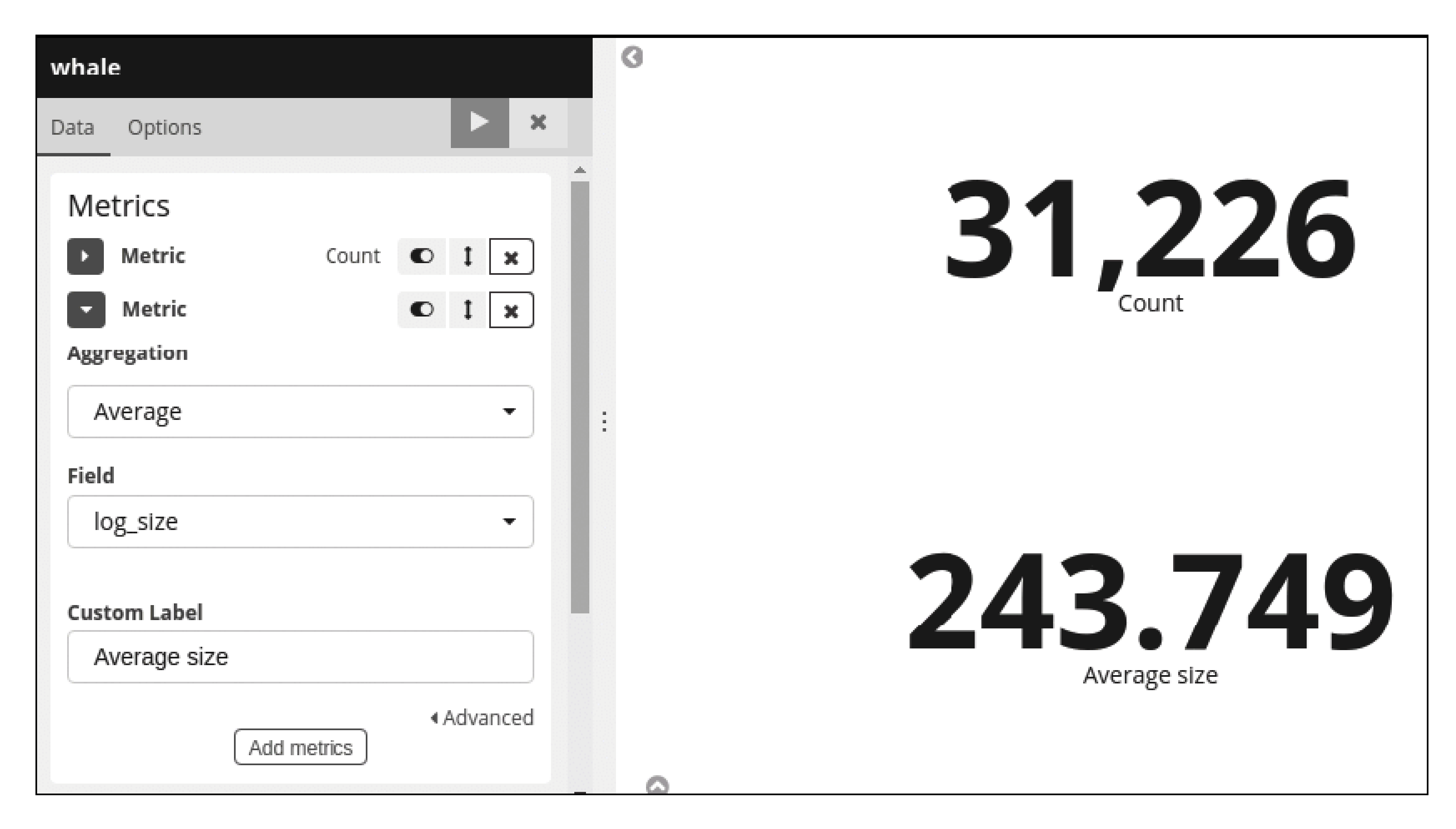


图 7.16 创建统计数值

每一种统计图表的数据均来源于界面左侧的聚合，单击 Aggregation 左侧的▶按钮，可以对该聚合的类型、字段、别名等属性进行设置。单击 Add metrics 按钮，可以添加新的聚合。单击执行按钮▶，可以在界面右侧显示出统计图表的效果。

在数值统计的创建过程中，对每一项可视化的聚合统计，只需指定单个数据项即可。开始创建时，默认提供了对文档中所有内容的数值统计（即 count 聚合）。图 7.16 中的第二个聚合是一个新加入的聚合，指定了聚合类型为平均值（Average），指定字段为日志长度（log_size），并为该聚合设置了别名 Average size。单击▶按钮即可看到右侧的效果。另外，单击上方的 Options 标签可以设置统计数据的字体大小。如果希望 Kibana 按数据中不同的数值分别进行统计，可以添加 Buckets 聚合，指定聚合类型为 Terms，在 Field（字段）下拉列表中指定要统计的字段，最后单击执行按钮▶即可。

下面创建一个标签云（该可视化方式可以直观地表现关键数据出现频度的情况）。在界面上方单击 New（新建）按钮，创建一个新的统计图表，选择 Tag cloud 选项，然后选择一个索引模式，进入统计图表创建界面。在左侧可视化数据的设置项中可看出，每一种聚合统计需要两个数据项才可以生成，其中，Metrics 聚合负责数值的计算操作，Buckets 聚合负责分类汇总操作。对于 Metrics 聚合部分，界面左侧默认给出了 Count 类型的聚合。在 Buckets 聚合中，只需单击左侧的下拉按钮，在 Aggregation 中选择 Terms，之后在 Field 中指定字段即可，如图 7.17 所示。聚合设置完毕之后，单击▶按钮即可生成标签云。在左侧数据设置面板上方单击 Options 标签，可以对标签云中的文字缩放模式、文字朝向、字体大小等属性

进行设置，如图 7.18 所示。生成的标签云如图 7.19 所示。

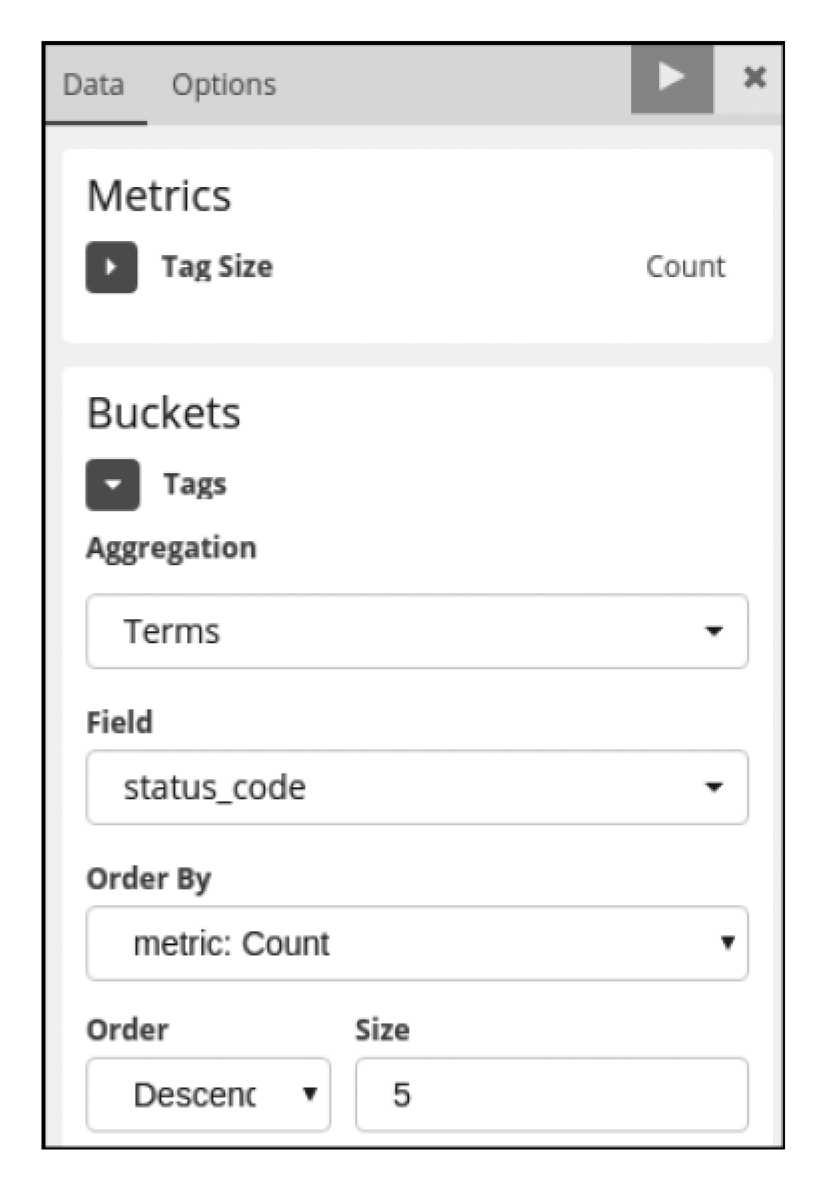


图 7.17 设置 Metrics 聚合

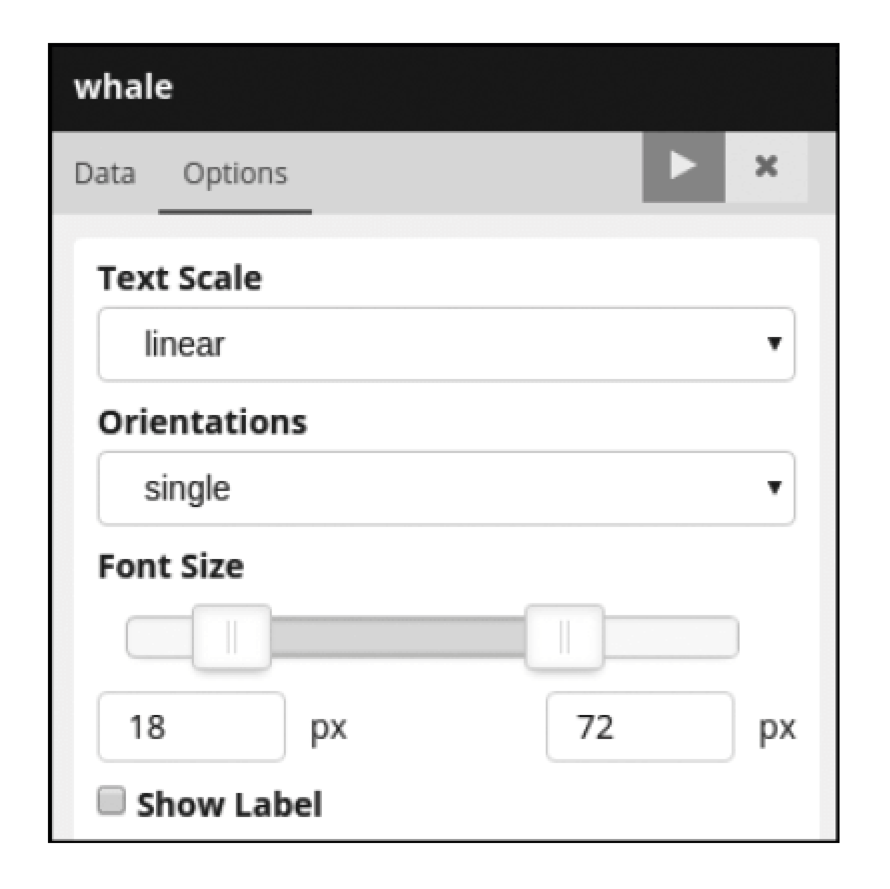


图 7.18 标签云设置选项

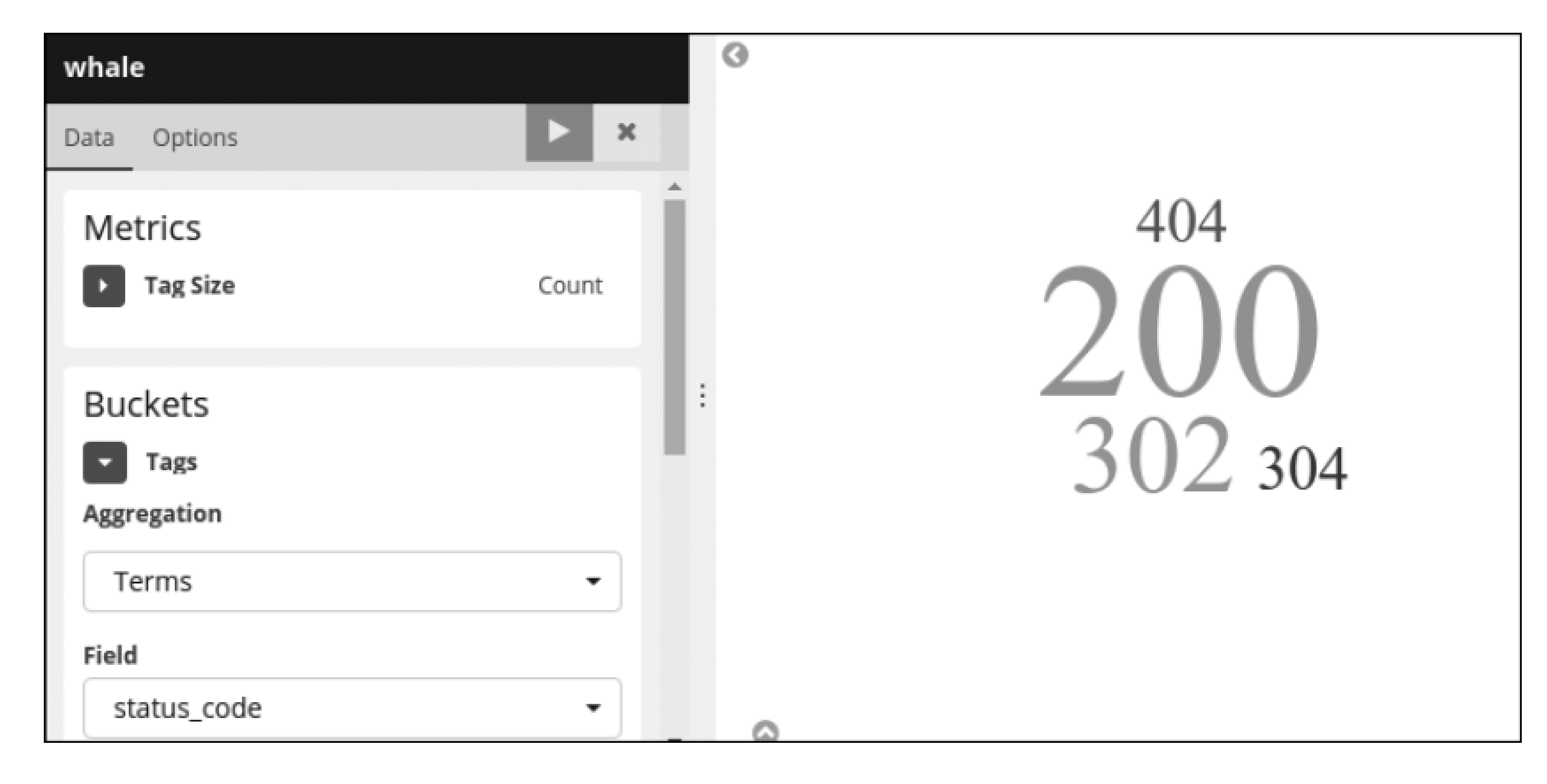


图 7.19 生成的标签云

7.6 使用 Dashboard 组件创建动态仪表板

Dashboard 组件提供了集中显示已保存的一系列可视化内容的功能。在其界面中可以整理各种统计图表并调整它们的大小。同其他程序生成的对象一样，动态仪表板也可以被

保存、加载、修改和分享。在 Kibana 前端界面左侧单击 Dashboard 导航按钮，即可跳转到 Dashboard 界面，如图 7.20 所示。本节对动态仪表板的创建、加载和分享等功能进行介绍。

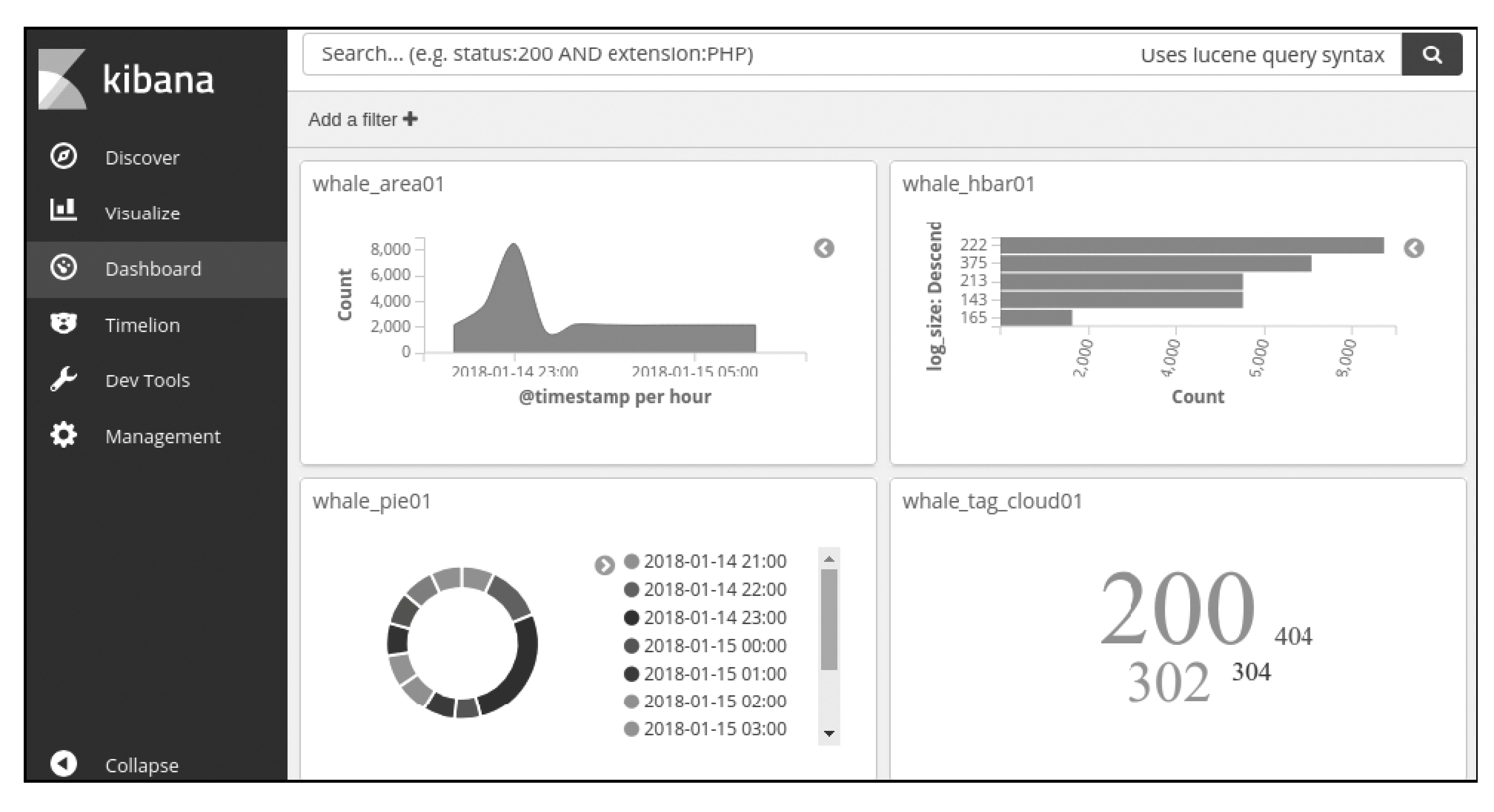


图 7.20　Dashboard 界面

7.6.1　创建新的动态仪表板

Dashboard 界面中提供了和 Discover 类似的时间选择器和检索输入框，可通过它们实现对 Elasticsearch 中的数据的预先查询，以满足仪表板上各种可视化统计图表的需要。在界面上方单击 Add 按钮（参见图 10.22），即可在已保存的可视化统计图表或检索结果中选择要添加到仪表板上的对象，该对象即出现在界面中，如图 7.21 所示。

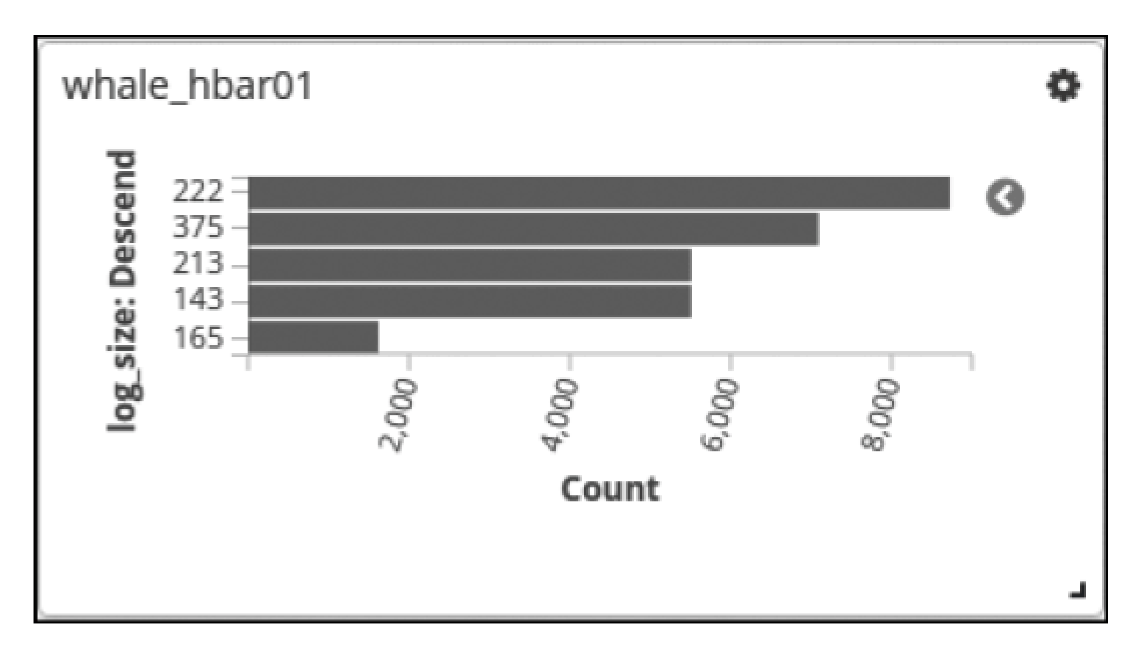


图 7.21　新加入仪表板的可视化内容

> Tips Kibana为Dashboard提供了亮色调和暗色调两种不同的主题风格，在界面上方单击Options按钮，勾选其中的Use dark theme选项，即可将界面转换为暗色调风格。该设置也可以在Management的高级设置界面中进行设置，将dashboard:defaultDarkTheme项设置为true即可转换为暗色调风格。

在加入新的仪表板后，可以对其执行编辑、移动、移除以及查看详细数据和改变大小等操作。在非编辑状态下，在任一个统计图表右上角按住按钮并拖曳，可以移动该统计图表到其他位置；单击左下角的按钮，可以查看构成该统计图表的各类统计数据，其中每个字段都提供了排序功能。要编辑已加入仪表板的可视化内容，需要先单击界面上方的Edit按钮方可进行编辑。在任一个统计图表右上角单击按钮，可以弹出可操作的选项。选择Edit visualization（编辑可视化内容）可以转到Visualize中，对该统计图表进行编辑；选择Delete from dashboard，可以将该可视化内容移除。按住并拖曳右下角的按钮，可以改变该可视化内容的大小。

如需保存一组动态仪表板，可以单击界面上方的Save按钮，输入当前动态仪表板的名称，如要将当前设置好的时间范围信息一并保存，可以勾选Store time with dashboard选项，最后单击Save按钮即可。

7.6.2 打开已保存的动态仪表板

在Dashboard界面中，要打开之前保存的动态仪表板，直接点选动态仪表板的名称即可。如果列表中的项目过多，可以在上方的过滤器中输入部分名称来进行过滤。

如果要对动态仪表板进行管理，例如执行导入、导出、删除等操作，可以转到Management界面中，单击Dashboards标签，即可对已保存的动态仪表板执行各种管理操作。

7.6.3 分享动态仪表板

动态仪表板的分享可以通过两种方式来完成。在Dashboard界面中打开一个动态仪表板，单击界面上方的Share按钮，可以通过直接复制Share面板中提供的URL链接，或复制<iframe>标记并嵌入网页前端代码中，来分享动态仪表板给其他用户。查看分享的动态仪表板的用户需要拥有Kibana的访问权限。

在Share面板右侧提供了分享动态仪表板当前状态的链接，这样的链接直接记录了仪表板中的状态信息，与原仪表板相互独立。考虑到有些浏览器在访问超长URL时可能会

出现错误，Share 面板中同时提供了复制相应的短链接的功能，访问时可以解析成原始的 URL。

7.7 使用 Timelion 组件创建时间线

Timelion 组件提供了基于时间线为不同数据来源显示统计曲线的功能。用户通过编写较为简单的查询表达式来检索基于时间序列的数据。针对较为复杂的计算问题，如“每位唯一身份的用户在一段时间内访问页面的数量”等，生成可视化的统计结果。在 Kibana 前端界面左侧单击 Timelion 导航按钮即可跳转到 Timelion 界面，如图 7.22 所示。本节对编写查询表达式生成时间线的方法进行介绍。

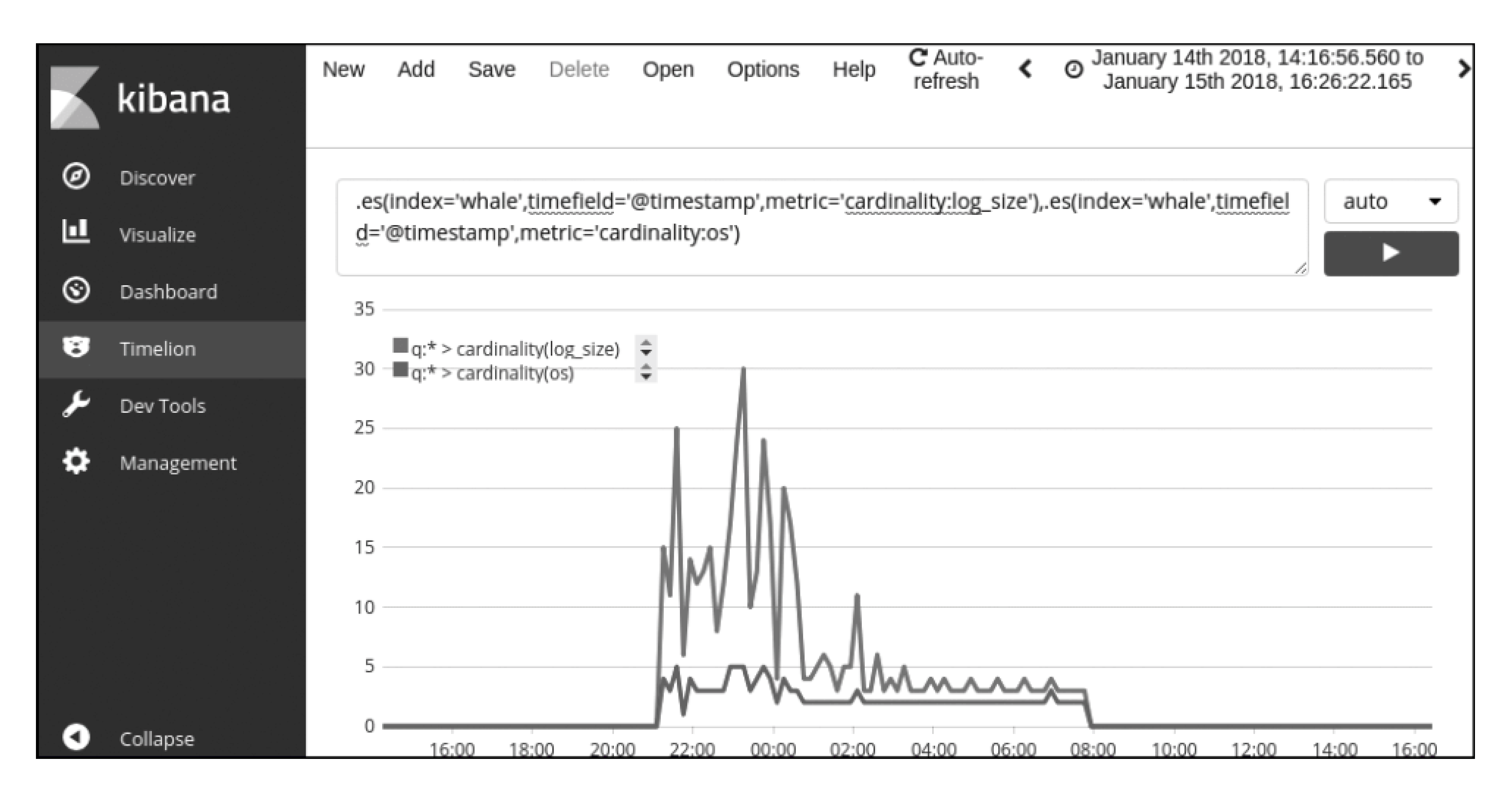


图 7.22 Timelion 界面

时间线的生成可通过在界面上方输入框中编写 Timelion 表达式来完成。Kibana 内置了 Timelion 表达式语言，在用户编写表达式的过程中，程序中会自动弹出相关提示，如图 7.23 所示。

Timelion 表达式语言以英文句点开头，以类似于高级编程语言中方法(函数)调用的格式来书写，例如图 7.23 中的.es()。如果要在时间线上添加另一条曲线，需要将两段英文句号开头的 Timelion 表达式分别写出来(以逗号隔开)。有些表达式是带有参数的(与高级编程语言类似，也是写在括号中的)，格式为“参数名='表达式'”，参数之间用逗号隔开。以.es()为例，如果要统计索引文件 whale 中 log_size 的数量，那么表达式应为.es(index='whale',timefield='timestamp',metric='cardinality:log_size')。其中，index 参数指定了数据来自索

```
.es(index='whale',timefield='@timestamp',m
```

.es() Pull data from an elasticsearch instance (Data Source)

Argument Name	Accepted Types	Information
q	*string, null*	Query in lucene query string syntax
metric	*string, null*	An elasticsearch metric agg: avg, sum, min, max, percentiles or cardinality, followed by a field. Eg "sum:bytes", "percentiles:bytes:95,99,99.9" or just "count"
split	*string, null*	An elasticsearch field to split the series on and a limit. Eg, "hostname:10" to get the top 10 hostnames
index	*string, null*	Index to query, wildcards accepted. Provide Index Pattern name for scripted fields and field name type ahead suggestions for metrics, split, and timefield arguments.

图 7.23　Timelion 表达式编写过程中的自动提示

引文件 whale；timefield 参数指定了带有时间戳的字段为 timestamp；metric 参数指定了具体的查询表达式是针对 log_size 字段，使用字段中的基数来统计。该统计生成的时间线如图 7.24 所示。

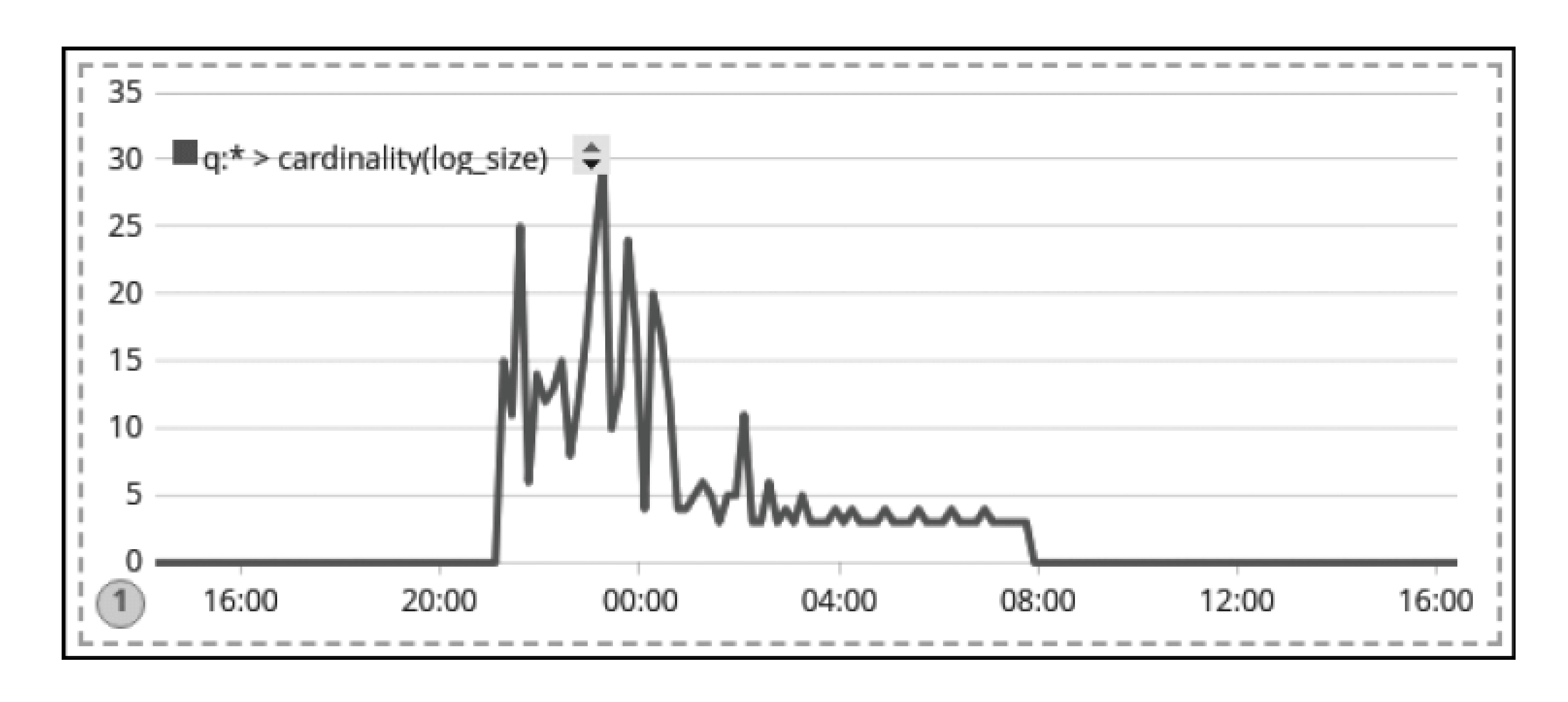


图 7.24　对 log_size 字段的统计结果时间线

Timelion 与 Dashboard 在使用方法上十分相似。在 Visualize 中也可以创建时间线。Timelion 支持对时间线的移除、改变位置等功能，添加了全屏放大显示的功能，也支持新建、保存、打开时间线等常用功能。除.es()表达式以外，还有更多 Timelion 表达式可供使用。限于篇幅，这里主要对 Kibana 中有关 Elasticsearch 的部分进行介绍，对于其他内容，读者可以自行实践。

7.8 使用 Dev Tools 执行命令行

Kibana 6.2 的 Dev Tools 界面中默认包含一个 Console 插件，该插件提供了一个通过 RESTful API 与 Elasticsearch 进行交互的界面。Console 的界面由两部分组成，左侧为 RESTful 命令行编辑器，右侧则为结果响应面板。在 Kibana 前端界面左侧单击 Dev Tools 导航按钮，即可跳转到 Dev Tools 中的 Console 插件界面，如图 7.25 所示。本节对 Console 的使用方法、快捷键、设置等进行介绍。



图 7.25 Dev Tools 中的 Console 界面

7.8.1 在 Console 中执行命令

在本书的第 2 章和第 3 章中，对于 Elasticsearch 的部分操作是在终端和可视化工具 head 中执行的。终端中使用的是 curl 语法。在 head 工具的复合查询页面中，可以直接在输入框中指定查询的索引和类型。而 Kibana 的 Console 中的命令行编辑器支持类似 curl 的语法格式。不同的是，Console 命令行头部直接由 HTTP 方法和查询的索引、类型等信息构成，命令的主体也不需要放进-d 后面的参数中。以查询索引文件 whale 中的全部文档信息为例，代码段 7.1 是采用了 curl 语法的查询命令。

代码段 7.1：在终端执行 curl 语法的查询命令

```
curl -H 'Content-Type: application/json' -XGET "http://localhost:9200/
_search" -d '
```

```
{
  "query": {
    "match_all": {}
  }
}'
```

代码段 7.2 则为 Console 中支持的语法。

```
代码段 7.2：在 Kibana 中以 Console 插件支持的语法执行查询命令
GET whale/_search
{
  "query": {
    "match_all": {}
  }
}
```

在编写命令行的同时，编辑器会为当前编写的代码自动设置缩进。写入具体内容时，编辑器中也会弹出代码提示列表。Console 支持多段命令批量执行，既可以将其中两段或更多的命令选中来批量执行，也可以将光标放置在其中一段上，单击右侧的执行按钮 ▶ 单独执行这一段。图 7.26 中包含 3 段命令，其中光标位置在第三段上，带有执行按钮和工具按钮的 Action 菜单跟随光标定位在第三段命令右侧。

```
PUT website
{⟺}

GET website/_search
{
  "query":{
    "match_all":{}
  }
}

PUT whale
{
  "mappings": {
    "_doc":{
      "properties": {
        "custom_ip":{
          "type": "keyword"
        },
        "@timestamp":{
          "type": "date",
          "format":"yyyy-MM-dd'T'HH:mm:ss"
        },
        "http_method":{
          "type": "keyword"
        },
        "uri":{
          "type": "keyword"
        },
```

图 7.26　编辑器可以执行多段命令，当前选中第三段

Action 菜单右侧的🔧按钮提供了 Auto indent(自动缩进)功能,光标所在段的代码格式不整齐时,将光标定位到该段代码头部,单击该按钮可以将代码梳理成缩进格式良好的代码。如果当前代码已经是格式良好的代码,再次单击🔧按钮,选择 Auto indent,可以将代码的主体压缩为单行,如图 7.27 所示。



图 7.27 压缩成单行的代码

Console 会自动保存最近的 500 条成功执行的命令行,单击界面上方的 History 按钮,界面中会显示历史记录面板。其中左侧是历史记录列表,选中其中一条并单击 Apply 按钮,可以将选中的命令添加到当前光标所在位置。单击 Clear 按钮,则会将历史记录清除。

7.8.2 Console 快捷键

Console 的编辑器支持快捷键功能。在编写命令行时使用相应的快捷键,有助于提高编写的效率。在这一点上,Console 比 head 使用起来更加方便。下面是一组用于常规编辑的组合键:

- Ctrl+I:将光标所在的命令梳理为缩进格式良好的代码,或将格式良好的代码压缩为单行。
- Ctrl+Space:弹出自动完成代码的提示列表。
- Ctrl+Enter:执行命令。
- Ctrl+↑/↓:将光标向上/向下跳到某段命令的开头/结尾。
- Alt+L:将光标所在的命令折叠为一个{⟺}图标,或从{⟺}图标恢复为原来的代码。

在 Console 的编辑器弹出代码提示的同时,另外 3 种按键可以用来进行提示框内部的操作。

- 方向键↑/↓:在提示列表中的不同项目之间进行选择。
- Tab 或 Enter:选择当前提示代码,自动完成编写。
- Esc:关闭提示框。

> Tips 单击 Dev Tools 界面右上角的 Help 按钮,界面中会显示出关于 RESTful API 和快捷键的使用说明。上述快捷键仅为常规键盘布局下的按键设置。使用说明中还提到了 Mac 键盘布局中的 Command 键和 Option 键,Mac 用户可阅读 Help 面板中的使用说明。

7.8.3 Console 设置

单击界面右上方的 Settings 按钮，界面中会显示 Console 设置面板。在其中可以设置界面中代码和结果的字号大小、是否允许换行以及自动完成编写的功能。

此外，在 Kibana 安装主目录下的 config 目录中的配置文件 kibana.yml 中可以对 Console 进行配置，下面是 3 个最常用的配置项：

- console.enabled：指定是否启用 Dev Tools 界面中的 Console 插件，默认为 true。
- server.host：指定 Kibana 可以接受来自哪些主机的访问，默认为 localhost，即只有本机可以访问。如果希望 Kibana 可被任何主机访问，可以将该设置项设为 0.0.0.0。
- server.port：指定 Kibana 运行在哪个端口，默认为 5601。

7.9 网站性能监控可视化应用实例

利用本书前面介绍的基于 Logstash 的网站操作日志，本节介绍基于 Kibana 的网站性能监控可视化应用实例。

7.9.1 概述

实验数据集来源自 Elasticsearch 中名为 whale 的索引文件，该索引文件的主要字段及其存储的实际数据示例如图 7.28 所示，这里的各个字段的含义如其名称的英文所示，在此不再赘述。本实例要用 Kibana 实现网站日志数据统计，以条形统计图、数据表、标签云、时间线和统计数值等形式展示出来。

```
"_index": "whale",
"_type": "_doc",
"_id": "s12kLWQBg1CkS_KXC9Uh",
"_version": 1,
"_score": null,
"_source": {
  "@timestamp": "2018-01-14T23:59:50",
  "http_method": "GET",
  "status_code": "200",
  "os": "Windows 10",
  "log_size": 143,
  "custom_ip": "172.16.1.154",
  "uri": "/commons/spider/getLongConnectionPort HTTP/1.1"
```

图 7.28　日志文件示例

7.9.2 使用 Visualize 实现可视化

目前在许多新媒体网站中均使用 Markdown 来快速创建美观、简洁的文字内容。拥有一个 Markdown 创建的文字板块可以使说明性文字排版整洁、一目了然。

在 Visualize 界面中选择 Markdown widget，并在左侧编辑区域填写 Markdown 格式的信息，如代码段 7.3 所示。

代码段 7.3：填写 Markdown 格式的文本信息

```
#Whale
###数据分析可视化展示
* 条形统计图
* 数据表
* 饼图
* 时间线
* 统计数值
```

然后单击▶按钮，即可在界面右侧得到如图 7.29 所示的可视化展示结果。单击屏幕上方的保存按钮，将该可视化结果保存起来。

接下来使用条形统计图来展示用户访问服务器的网络状态码 status_code 字段的数据。在 Visualize 界面中选择 Vertical Bar Chart 选项，在界面左侧已有的 Y-Axis 中的 Aggregation 下拉列表中选择 Percentile Ranks 选项来统计 status_code 不同数值的占比。在 Values 中添加 4 个数值，分别为 200、302、304 和 404（这只是当前 whale 索引文件中的内容，读者需要根据自己的数据文件的实际内容来确定）。下面添加 X-Axis 聚合，在 Aggregation 下拉列表中选择 Date Histogram，设置 Field 为@timestamp，设置 Interval（指定间隔）为自动，单击▶按钮后，界面右侧会生成如图 7.30 所示的柱状图。单击屏幕上方的保存按钮，将该可视化结果保存起来。

Whale

数据分析可视化展示

- 条形统计图
- 数据表
- 饼图
- 时间线
- 统计数值

图 7.29　Markdown 部件可视化结果

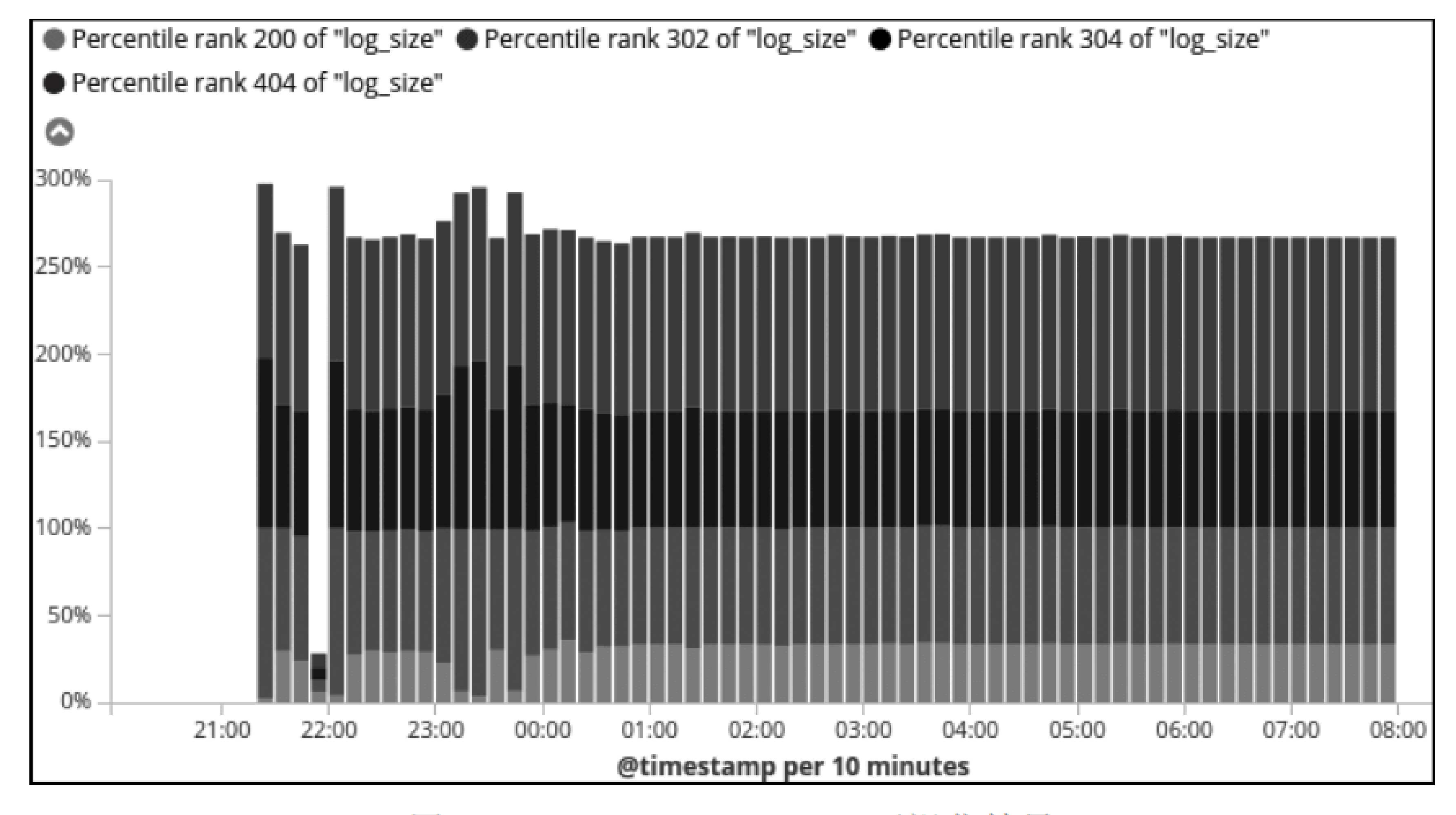


图 7.30　Vertical Bar Chart 可视化结果

下面使用数据表对用户访问服务器的网络状态数据进行具体数值方面的统计。在 Visualize 界面中选择 Data Table 选项，左侧靠上的 Metric 保持 Count 聚合不变，在下方添加一个 Split Rows 聚合，Aggregation 选择 Terms，Field 选择 status_code，Order By 选择 Term，Order 选择 Desending，Size 填写 4（数据中仅包含 4 种状态码）。将 Options 标签中的 Per page 设置为 4，以避免表格下方出现大面积空白。单击▶按钮，界面右侧将出现如图 7.31 所示的数据表。单击屏幕上方的保存按钮，将该可视化结果保存起来。

status_code: Descending	Count
404	3
304	200
302	8,731
200	22,292

Export: Raw Formatted

图 7.31　Data Table 可视化结果

不同的用户在访问服务器时使用的操作系统各不相同，这里使用饼图对用户使用的操作系统名称进行统计并展示各种操作系统的占比。在 Visualize 界面中选择 Tag Cloud 选项。左侧靠上的 Tag Size 保持 Count 聚合不变。在下方的 Buckets 聚合中，设置聚合类型为 Term，字段指定 os，即统计访问的操作系统版本。下方的 Size 设为 6（数据中仅包含 6 种操作系统版本），其他项保持默认设置即可。单击▶按钮，界面右侧将出现如图 7.32 所示的标签云。单击屏幕上方的保存按钮，将该可视化结果保存起来。



图 7.32　Tag Cloud 可视化结果

下面使用时间线来统计和展示服务器日志的分时数据量以及用户访问服务器的网络状态码走势。在 Visualize 界面的输入框中输入关于日志记录长度字段 log_size、网络状态码字段 status_code 以及操作系统版本字段 os 的查询表达式，如代码段 7.4 所示。

代码段 7.4：在 Timelion 界面填写查询表达式

```
.es(index='whale',timefield='timestamp',metric='cardinality:log_size'),
.es(index='whale',timefield='timestamp',metric='cardinality:status_code'),
.es(index='whale',timefield='@ timestamp',metric='cardinality:os')
```

单击▶按钮，查看界面中时间线是否出现起伏。如果时间线仍为直线，可以尝试单击界面右上角的时间选择器，选择一个更大的时间范围。时间线出现起伏之后，可以按下鼠标左键拖曳，滑过起伏的部分，使得时间轴上的时间段缩小，使起伏部分占满屏幕。在操作成功的情况下，界面右侧将出现如图 7.33 所示的曲线。单击屏幕上方的保存按钮，将该可视化结果保存起来。

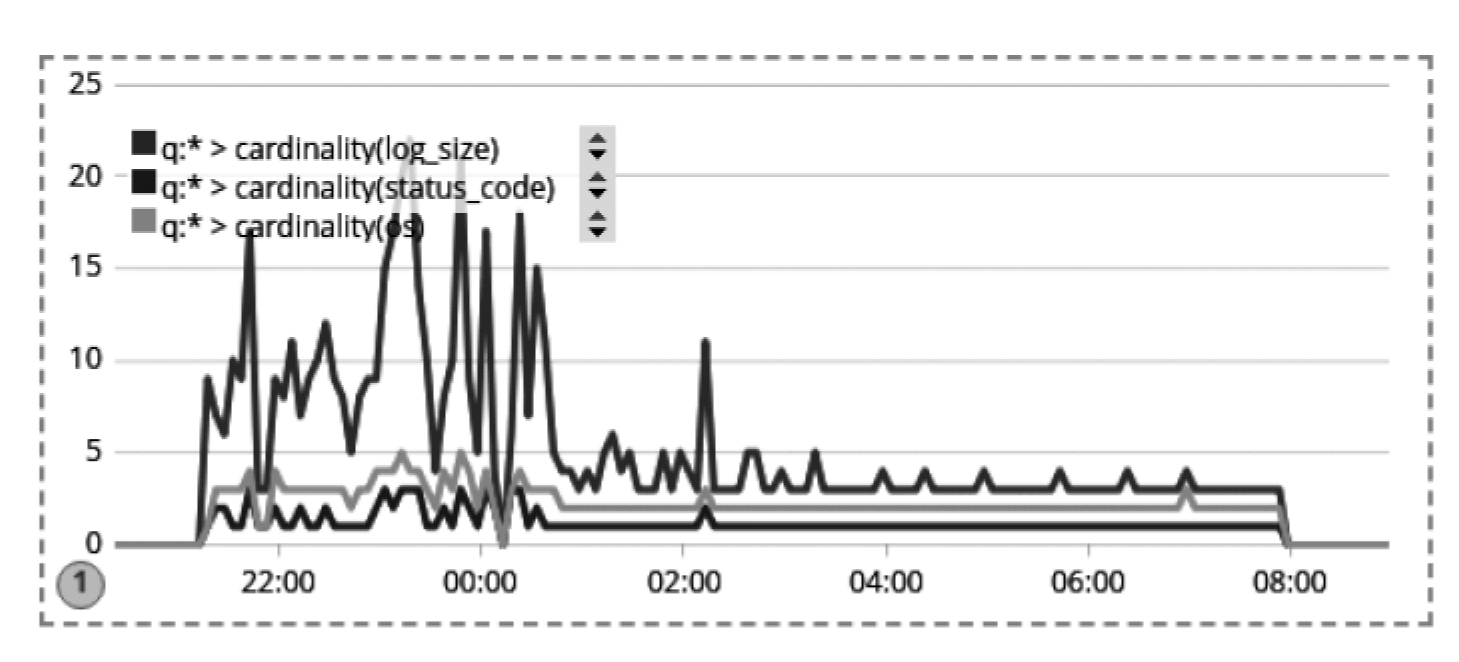


图 7.33 Timelion 可视化结果

最后使用数据统计功能统计服务器日志分时数据量的平均值和最大值。在 Visualize 界面中选择 Metrics 选项，将左侧已有的聚合展开，Aggregation 选择 Average，Field 选择 log_size，然后添加一个新的 Metric，Aggregation 选择 Max，Field 仍选择 log_size。在下方的 Buckets 聚合中，单击 Split Group 按钮添加新聚合，Aggregation 选择 Date Histogram，Field 选择@timestamp，Interval 选择 Daily。在 Options 标签中，展开 Style 部分，将字号修改为 18pt。单击▶按钮，界面右侧将出现如图 7.34 所示的统计数值。单击屏幕上方的保存按钮，将该可视化结果保存起来。

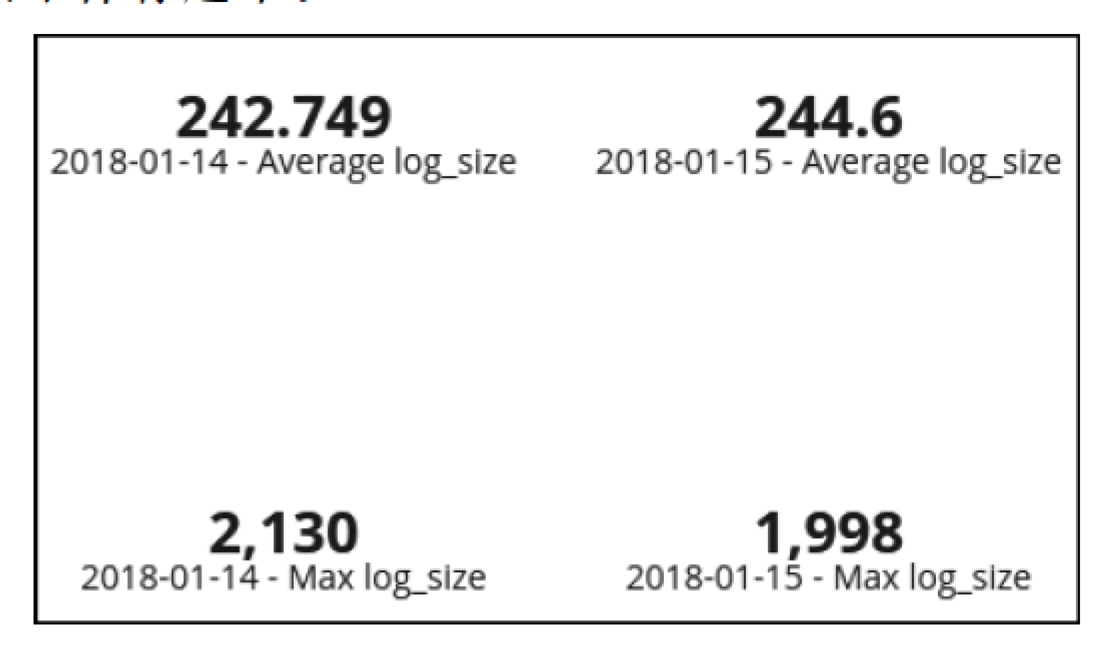


图 7.34 Metric 可视化结果

7.9.3 使用 Dashboard 整合可视化结果

单击 Kibana 界面左侧的 Dashboard 导航按钮，进入 Dashboard 界面。单击界面上方的 Add 按钮，逐个将上面创建好的 6 种可视化统计图表添加到动态仪表板中，并通过拖曳每一种统计图表上的标题栏和 ◢ 按钮来调整其位置和大小，Dashboard 将为每一个面板自动调整位置和大小，使所有面板在界面中对齐。全部操作完成后，单击屏幕上方的保存按钮，勾选保存时间的选项，保存该动态仪表板。可视化展示的最终效果如图 7.35 所示。

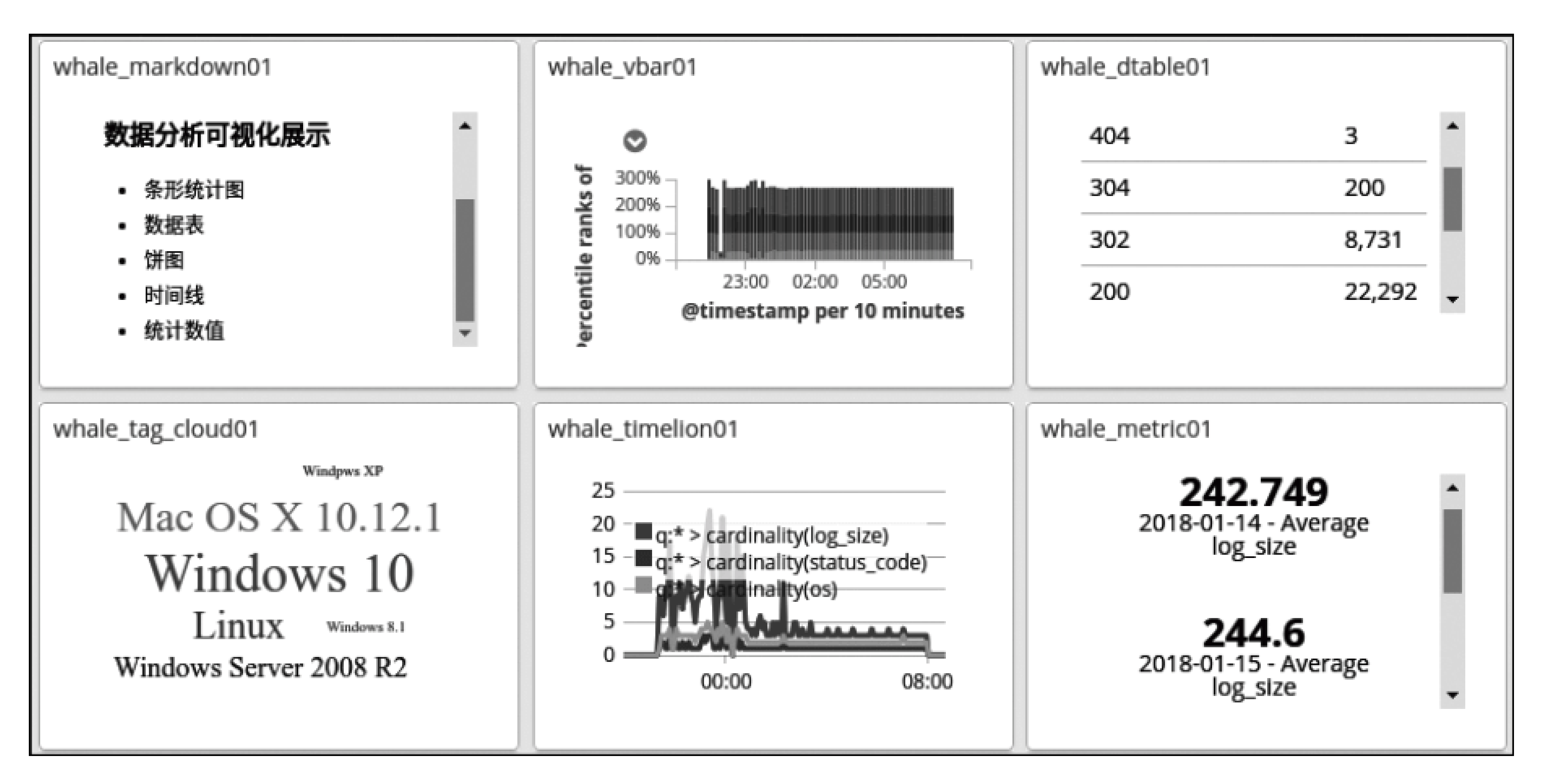


图 7.35　可视化展示最终效果

单击界面上方的 Edit 按钮，再单击 Options 按钮，勾选 Use dark theme 选项，可以将界面切换至暗色调主题。切换后的动态仪表板展示效果如图 7.36 所示。

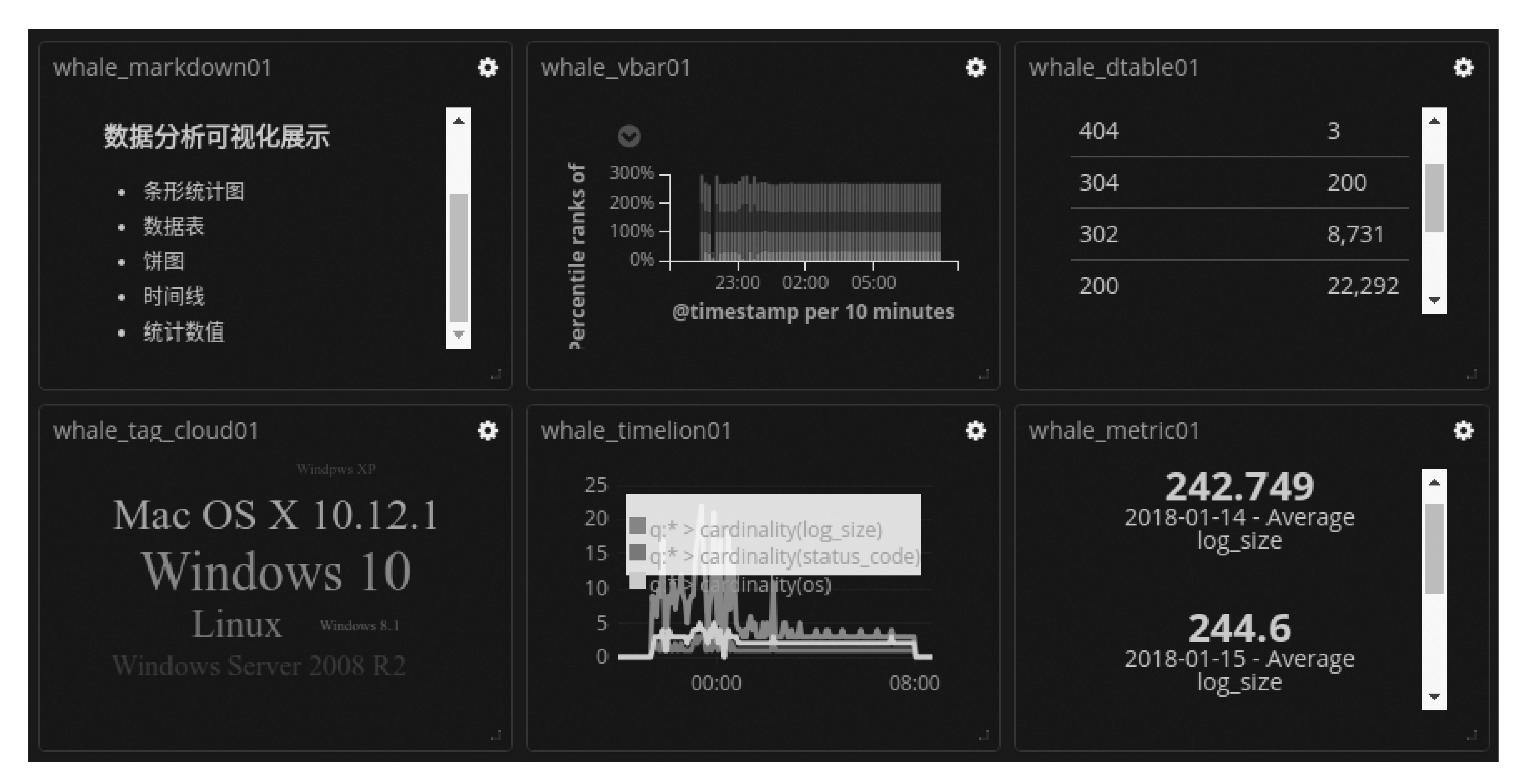


图 7.36　暗色调主题的可视化展示效果

7.10 扩展知识与阅读

信息可视化技术是利用计算机实现对抽象数据的可视化表示，以此来增强人们对抽象信息的感知。它不仅在揭示信息资源的广度与深度上有很大优势，而且能够将隐藏在信息资源内部复杂的、抽象的信息以直观的方式呈现给用户。它可以利用人们对可视模式快速识别的能力，将数据信息转化为视觉形式，形象化地揭示数据深层次的联系、趋势等。近年来，信息可视化的研究已经引起了科研人员的密切关注，欧美国家启动了相应的科研计划，在理论模型和应用技术方面已取得了较大进展，并在 IEEE 系列会议、信息可视化和人机交互相关会议（如 SIGGRAPH、ACM PVG、ACM SIGCHI 等）上发表了一些重要的研究成果。为了建立数据的可视映射，有的文献提出了可视化参考模型来描述原始数据、数据表格、可视化架构和视图之间的转换关系以及用户根据任务通过人机界面进行的数据变换、可视化映射、视图变换等操作。

已有的一些可视化工具软件包括：

（1）D3. js。它是一个强调网页标准的用来创建数据可视化的 JavaScript 库，使用 HTML、SVG 和 CSS，可以让用户以数据驱动的方式操作 DOM，可绑定任意数据到 DOM，可创建交互式 SVG 条形图，可从数据集里产生 HTML 表格，并提供多种组合和插件来增强兼容性。

（2）Gephi。它是一个能在 Windows、Linux 和 Mac OS 系统上运行的开源程序，用于可视化地探索数据，其插件更加个性化。

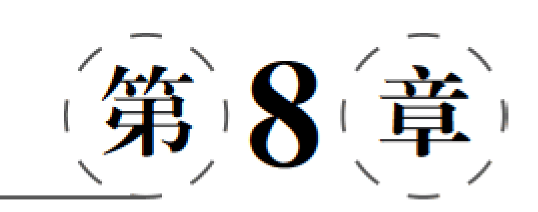
（3）R Project。它是在 UNIX、Windows 和 Mac OS 上运行的统计计算软件，可用于数据处理、计算和图表展示，是用于即时分析的集成工具。

（4）Prefuse 软件包和 Prefuse Flare 工具包。前者为 JavaScript 提供了可视化框架，而后者为 Adobe Flashplayer 等提供了可视化和动画化工具，能完成数据建模、数据交互化和可视化，对各种视觉布局进行数据结构优化，支持动画、动态搜索和数据库连接。

基于 Kibana 的信息可视化能够完成对信息的显示处理，比较简单和易于上手，这也是 Kibana 广为流行的原因之一。除 Kibana 外，也有一些免费开源方案可用来支撑数据可视化应用，如上述的 D3. js、Gephi 等。可视化的工具还包括 Excel、Google Spreadsheets、Tableau 等。在编程工具方面，R 语言是一个用于统计学计算和绘图的语言，也能在数据的可视化表示方面发挥很大的作用（周苏，2016）。对学术界来说，为了建立数据的可视映射，需要建立可视化参考模型，用来描述原始数据、数据表格、可视化架构和视图之间的转换关系以及用户根据任务通过人机界面进行的数据变换、可视化映射、视图变换等操作。相关资料可以参阅文献（周宁，2005）。

7.11 本章小结

Kibana 是为 Logstash 和 Elasticsearch 等组件提供的日志可视化分析工具，可使用它对日志进行高效的可视化、分析等各种操作。在基于 Logstash 日志的数据基础上，本章对 Kibana 的数据检索、可视化统计图表、动态仪表板、时间线和开发工具的一般配置进行了概述，并以 Logstash 日志为例，介绍了基于 Kibana 的可视化实现。通过 Kibana 提供的交互式界面，可以很快定位到异常事件或者将其查找范围缩小。



基于 X-Pack 的系统运行监控

> X-Pack is an Elastic Stack extension that bundles security, alerting, monitoring, reporting, and graph capabilities into one easy-to-install package. While the X-Pack components are designed to work together seamlessly, you can easily enable or disable the features you want to use. Prior to Elasticsearch 5.0.0, you had to install separate Shield, Watcher, and Marvel plugins to get the features that are bundled together in X-Pack. With X-Pack, you no longer have to worry about whether or not you have the right version of each plugin, just install the X-Pack for the Elasticsearch version you're running, and you're good to go!
>
> https://www.elastic.co/guide/en/x-pack/current/xpack-introduction.html

在 Elasticsearch 2.x 及更早的版本中，可以集成 head、Marvel、Shield、Watcher、Reporting、Graph 等多种插件。升级到 5.0 版本之后，第三方 head 工具不再以插件形式集成(有关 head 的内容详见第 2 章的介绍)。而对于 Marvel、Shield、Watcher、Reporting、Graph 等插件，Elastic 公司推出了新的 X-Pack 插件取而代之。X-Pack 插件包含了 Security(即以前版本中的 Shield)、Monitoring(即以前版本中的 Marvel)、Alerting and Notification 以及 Graph 等功能，本章基于 X-Pack 6.2.4，对相关插件的使用方法进行介绍。

8.1 X-Pack 概述

在 Elasticsearch 5.0 版本之前，各种插件，如 Shield、Watcher、Marvel 等，需要分别安装和使用。在 Elasticsearch 5.0 版本发布之后，Elastic 公司推出了 X-Pack，将上述插件全部集成在了独立的 X-Pack 中。X-Pack 是 Elastic Stack 的扩展，将 Security、Alerting、Monitoring、Reporting、Graph、Machine Learning 等插件集成到同一个软件包中，安装也比

较简便。X-Pack 可以与 Elasticsearch 和 Kibana 无缝对接、协同工作，其中的部分组件也可以根据实际需要随时启用或禁用。X-Pack 的出现，消除了各插件因版本不一致等而产生的各种问题，用户只需使 X-Pack 的版本与 Elasticsearch 保持一致，即可正常使用。

8.2 安装 X-Pack

使用 X-Pack，需要 Elasticsearch 和 Kibana 均已完成安装。它的安装方法如下：

首先下载 Elastic 公司发布的 X-Pack 软件包，可以通过两种方法来下载安装：

（1）通过在终端输入命令来完成。进入 Elasticsearch 的 bin 目录，并在终端执行命令./elasticsearch-plugin install x-pack，程序将会自动进行安装。

（2）访问 Elastic 官网 https://artifacts.elastic.co/downloads/packs/x-pack/x-pack-6.2.4.zip 下载软件包。注意，链接中的 6.2.4 是软件包的版本号，该版本号应该与当前已安装的 Elasticsearch 版本一致（读者应根据实际情况选择相应的软件包，下同）。下载完毕后，在 Elasticsearch 和 Kibana 未启动的状态下进行安装。不要将 ZIP 文件解压，直接进入 Elasticsearch 的 bin 目录，在终端执行以下命令：

```
./elasticsearch-plugin install file:///软件包的绝对路径/x-pack-6.2.4.zip
```

安装过程中，X-Pack 会请求附加权限，以便 Watcher 可以发送电子邮件通知，这可能会带来某些安全隐患。安装 Elasticsearch 时的终端界面如图 8.1 所示。

```
cy@cy-N53SN:/usr/elasticsearch-6.2.4/bin$ ./elasticsearch-plugin install file://
/usr/x-pack-6.2.4.zip
-> Downloading file:///usr/x-pack-6.2.4.zip
[=================================================] 100%
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@     WARNING: plugin requires additional permissions     @
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
* java.io.FilePermission \\.\pipe\* read,write
* java.lang.RuntimePermission accessClassInPackage.com.sun.activation.registries
* java.lang.RuntimePermission getClassLoader
* java.lang.RuntimePermission setContextClassLoader
* java.lang.RuntimePermission setFactory
* java.net.SocketPermission * connect,accept,resolve
* java.security.SecurityPermission createPolicy.JavaPolicy
* java.security.SecurityPermission getPolicy
* java.security.SecurityPermission putProviderProperty.BC
* java.security.SecurityPermission setPolicy
* java.util.PropertyPermission * read,write
See http://docs.oracle.com/javase/8/docs/technotes/guides/security/permissions.h
tml
for descriptions of what these permissions allow and the associated risks.

Continue with installation? [y/N]y
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@        WARNING: plugin forks a native controller        @
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
This plugin launches a native controller that is not subject to the Java
security manager nor to system call filters.

Continue with installation? [y/N]y
-> Installed x-pack with: x-pack-logstash,x-pack-deprecation,x-pack-security,x-p
ack-ml,x-pack-watcher,x-pack-upgrade,x-pack-core,x-pack-graph,x-pack-monitoring
cy@cy-N53SN:/usr/elasticsearch-6.2.4/bin$
```

图 8.1　为 Elasticsearch 安装 X-Pack 插件

在 Kibana 中安装 X-Pack 插件时，需进入 Kibana 的 bin 目录，在终端执行以下命令：

```
./kibana-plugin install file:///软件包的绝对路径/x-pack-6.2.4.zip
```

安装过程的终端界面如图 8.2 所示。

```
cy@cy-N53SN:/usr/kibana-6.2.4-linux-x86_64/bin$ ./kibana-plugin install file:///
usr/x-pack-6.2.4.zip
Attempting to transfer from file:///usr/x-pack-6.2.4.zip
Transferring 309419696 bytes....................
Transfer complete
Retrieving metadata from plugin archive
Extracting plugin archive
Extraction complete
Optimizing and caching browser bundles...
Plugin installation complete
cy@cy-N53SN:/usr/kibana-6.2.4-linux-x86_64/bin$
```

图 8.2 为 Kibana 安装 X-Pack 插件

> Tips “file:///路径”的写法是一种特定的协议书写格式，由“file://”和“/路径”组成。在上面的命令中提供的是软件包的绝对路径。

要对 X-Pack 进行更新，应先卸载旧版本的插件，再重新安装最新版本的插件。卸载 X-Pack需要在终端执行如下命令：

```
bin/elasticsearch-plugin remove x-pack
bin/kibana-plugin remove x-pack
```

卸载插件后，重启 Elasticsearch 和 Kibana 即可。

8.3 Security 插件与安全性

X-Pack 中的 Security 插件可以实现集群安全和数据的密码保护，其中还包含多种高级安全机制，如通信加密、基于角色的访问控制、IP 过滤以及客户端访问的安全机制等。本节对加入 X-Pack 插件后 Kibana 的部分安全功能以及与安全集群交互操作的方法进行介绍。

8.3.1 身份验证机制与用户管理

为实现集群安全，需要集群中的每一个节点均安装了对应版本的 X-Pack 插件。默认情况下，用户认证机制是开启的。安装 X-Pack 插件后，用户需要先为 X-Pack 内置的用户账户设置密码方可正常登录。

X-Pack 中的 Security 插件内置 3 个账号：超级管理员账号名为 elastic，拥有对集群的完全访问权限；Kibana 管理员账号名为 kibana，用于控制 Kibana 对 Elasticsearch 进行连接

和交互；Logstash 管理员账号名为 logstash_system，用于控制 Logstash 存储 Elasticsearch 中来自 Monitoring 插件的索引数据。

设置内置用户密码最简单的方式是使用 setup-passwords 工具执行批量设置，但该操作是一次性的，如要二次修改密码，则只能使用 Kibana 中的 Management 程序，或使用X-Pack 提供的 Change Password API 来执行。启动 Elasticsearch，然后进入 Elasticsearch 目录下的 bin/x-pack 目录，执行如下终端命令来批量设置内置用户的密码：

```
./setup-passwords interactive
```

此时程序将提示用户对 3 个用户分别设置和确认密码，如图 8.3 所示。

```
cy@cy-N53SN:/usr/elasticsearch-6.2.4/bin/x-pack$ ./setup-passwords interactive
Initiating the setup of passwords for reserved users elastic,kibana,logstash_sys
tem.
You will be prompted to enter passwords as the process progresses.
Please confirm that you would like to continue [y/N]y


Enter password for [elastic]:
Reenter password for [elastic]:
Enter password for [kibana]:
Reenter password for [kibana]:
Enter password for [logstash_system]:
Reenter password for [logstash_system]:
Changed password for user [kibana]
Changed password for user [logstash_system]
Changed password for user [elastic]
cy@cy-N53SN:/usr/elasticsearch-6.2.4/bin/x-pack$
```

图 8.3　为 3 个内置用户设置密码

接着，在 Kibana 配置文件中加入已设置的账号和密码信息，进入 Kibana 目录下的 config 目录，编辑 kibana.yml 配置文件，将其中关于 Elasticsearch 登录的配置修改为如下配置信息：

```
elasticsearch.username: "elastic"
elasticsearch.password: "elastic"
```

其中，第一行指定用户名 elastic，第二行为 elastic 用户设置登录密码。

启动 Kibana，然后在浏览器中访问 Kibana，将不会直接显示 Kibana 界面，而是先显示用户登录界面，如图 8.4 所示。用户需要输入登录信息，完成身份认证后方可进入 Kibana 的操作界面。

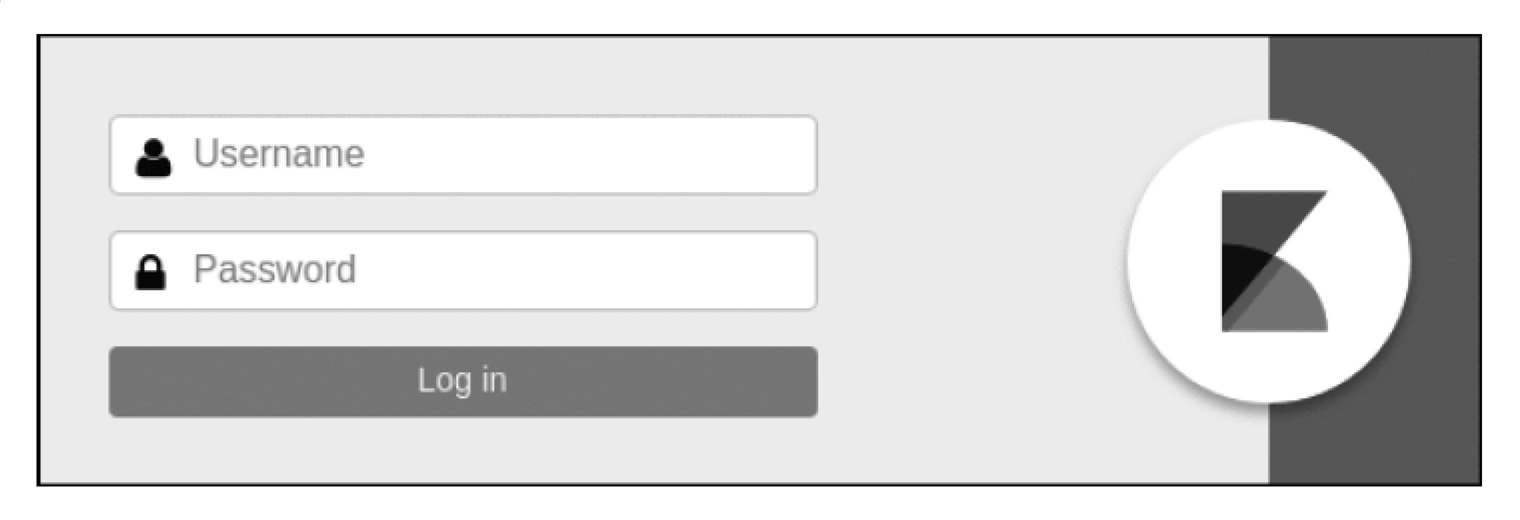


图 8.4　Kibana 的登录界面

如要使用 Change Password API 来修改用户密码，运行如代码段 8.1 所示的终端命令即可，其中的＜username＞表示要修改密码的用户名。

代码段 8.1：修改用户密码

```
curl -H 'Content-Type: application/json' -XPOST -u elastic:elastic
'localhost:9200/_xpack/security/user/<username>/_password' -d '{
  "password" : "new_password"                    //设置新密码
}'
```

在 Kibana 前端界面中也提供了修改用户密码的功能。登录 Kibana 后，单击左侧导航栏下方的用户名，界面右侧即显示出对当前用户的管理功能，在其中可以设置用户的电子邮件地址以及修改密码。单击 Change Password 按钮，即可在弹出的文本框中修改密码，如图 8.5 所示。

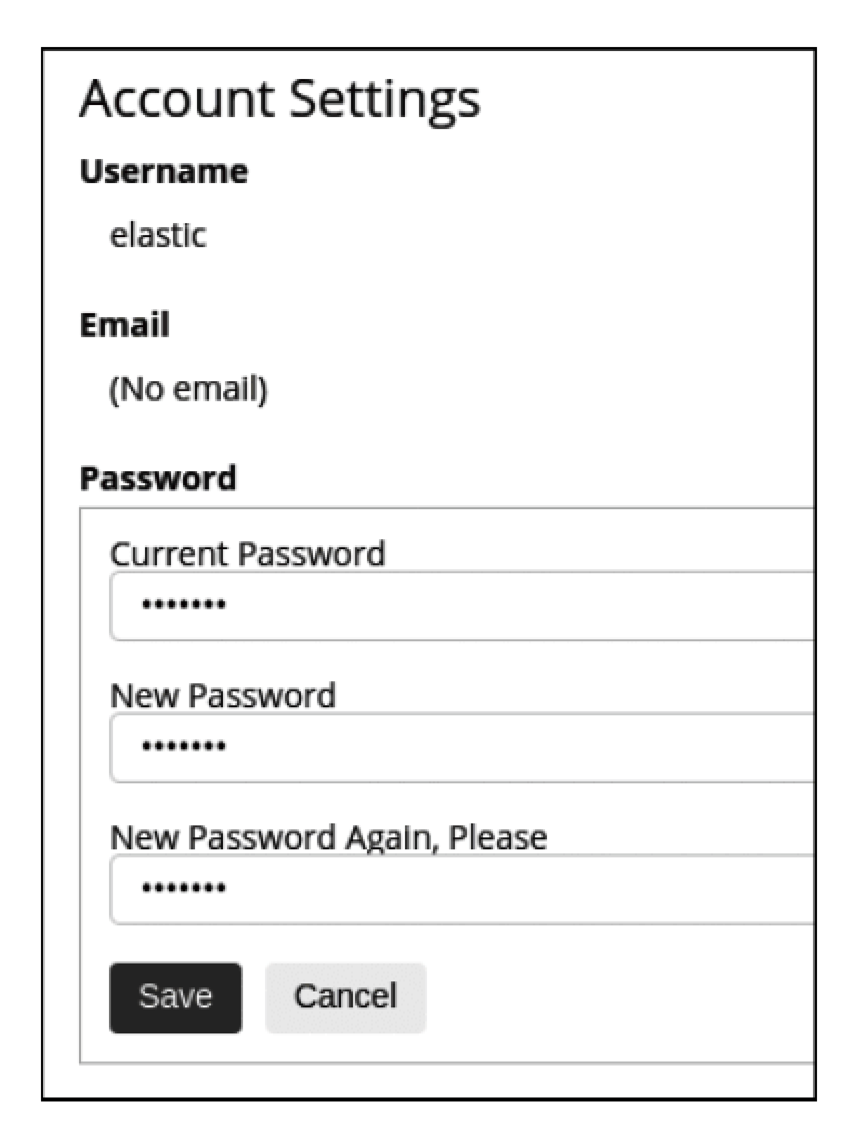


图 8.5　设置用户信息和修改密码

> Tips 在为内置账号 elastic 修改密码的同时，需要在 Kibana 安装主目录下的 config 目录中的配置文件 kibana.yml 中进行相应的配置，即在文件中修改配置项 elasticsearch.password：“新密码”。

安装 X-Pack 插件后，Management 程序中含有 Security、Elasticsearch、Kibana 和 Logstash 4 种管理选项。管理员账户可以通过这里提供的功能来管理各类功能和用户。例

如，在 Security 管理选项中单击 Users 链接，即可跳转至系统中的用户管理页面，查看系统中所有用户的列表。界面上方的 Search 输入框可以用来对现有用户进行过滤查找，单击其中一个用户名，即可转到该用户的详细信息页面，其中包括用户名、密码修改链接、用户角色等信息。

要添加新用户，可以在用户管理页面中单击 Create user 按钮，在新用户创建页面中填写用户名、密码、确认密码、用户全名和电子邮件地址，并单击下方的 Add a role 为新用户设置权限（可以添加多个权限），最后单击 Save 按钮保存即可，如图 8.6 所示。已添加的新用户将出现在用户列表中，单击任意一个用户名即可查看该用户的详细信息和权限，也可以修改用户信息。勾选用户并在界面右上角单击 Delete 按钮可以将该用户删除。

图 8.6　添加新用户并设置权限

8.3.2　匿名访问

当程序接收的用户请求中不包含任何用户认证信息时，该请求就会被识别为匿名请求。默认情况下，程序会拒绝匿名请求，并返回包含状态码 401 的认证错误信息。如果要启用匿名访问的功能，需要在 Elasticsearch 安装主目录下的 config 目录下的配置文件 elasticsearch.yml 中添加如代码段 8.2 所示的配置信息，这一项配置相当于对没有身份认证的部分操作给予认可。

代码段 8.2：通过 elasticsearch.yml 配置文件启用匿名访问

```
xpack.security.authc:
  anonymous:
    username: anonymous_user
    roles: kibana_user, transport_client            //指定用户访问权限
    authz_exception: true
```

其中，username 指定了匿名用户的用户名，如果不指定，则默认用户名为 _es_anonymous_user；roles 指定了匿名用户的访问权限，如果不指定，则一切匿名访问将会被拒绝并返回认证错误信息；authz_exception 指定了当匿名用户执行权限之外的请求时的处理策略（默认为 true，此时，如果匿名用户不具备当前操作所需的权限，程序将返回状态码 403 并且不会提示用户进行身份认证；如将该项配置为 false 且匿名用户执行了权限之外的操作，程序会返回状态码 401，并提示用户进行身份认证方可执行操作）。

8.3.3　基于域的用户认证

Security 插件内置了 5 种称为域的用户身份认证服务，分别为 native、ldap、active_directory、pki 和 file。下面分别进行介绍。

- native：用户信息存储在专门的 Elasticsearch 索引中，该域支持以用户名和密码的方式来验证用户身份，如果 elasticsearch.yml 配置文件中没有对该域的其他配置，那么该域默认可用。
- ldap：通过使用外部轻量目录访问协议来验证用户身份的域。在此域中，用户可以输入用户名、密码等认证信息来进行身份验证。
- active_directory：通过使用外部活动目录服务器来验证用户身份的域。在此域中，用户可以输入用户名、密码等认证信息来进行身份验证。
- pki：通过使用公钥基础设施来验证用户身份的域。这样的域可以与 SSL/TLS 协同工作，并通过客户端的 x.509 证书的标识名来识别用户。
- file：用户信息存储在 Elasticsearch 集群每个节点的文件中。该域支持以用户名、密码等认证信息来进行身份验证。

除以上 5 种域之外，Security 插件还支持自定义域。不同的域则通过 elasticsearch.yml 配置文件组织在一个域链当中。代码段 8.3 展示了配置文件中的配置信息。

代码段 8.3：elasticsearch.yml 中的域链

```
xpack.security.authc:
  realms:
    file:
      type: file
      order: 0                    #第 1个验证
    native:
      type: native
      order: 1                    #第 2个验证
    ldap1:
      type: ldap
      order: 2                    #第 3个验证
      enabled: false
      url: 'url_to_ldap1'
      …
    ldap2:
      type: ldap
      order: 3                    #第 4个验证
```

```
        url: 'url_to_ldap2'
        ...
      ad1:
        type: active_directory
        order: 4                    #第 5个验证
        url: 'url_to_ad'
      authz_exception: true
```

在这段配置信息中，type 表示域认证的类型，order 表示不同的域认证的顺序，enabled 表示是否在域链中启用该域的认证。列表的顺序决定了访问域的顺序。认证过程中，Security 逐个对链中的域进行访问并尝试认证。一旦某次认证通过，那么用户的操作就会被认为通过了身份认证；如果当前域未通过认证，那么程序将访问下面的域，直到该段配置信息末尾；如果全部域均未通过认证，那么用户操作就会被认定为身份认证失败，程序将返回包含状态码 401 的认证错误信息。下面以 native 域用户身份认证为例，对域的配置进行介绍。

在 elasticsearch.yml 配置文件的 xpack.security.authc.realms 命名空间中配置 native 类型。代码段 8.4 展示的是将 native 域设为链中首个认证域的方法。

```
代码段 8.4：配置 native 域
xpack.security.authc.realms:          //命名空间
    realm1:
        type: native                  //指定域类型为 native
        order: 0                      //指定认证顺序
        enabled: true                 //启用
```

配置完成后重启 Elasticsearch，该域配置即生效。在配置信息中，除上述的 type、order、enabled 属性外，还有如下 3 个属性可供配置。

（1）cache.ttl：指定用户信息缓存的生命周期时长。在指定的时间内，用户的信息会被缓存，该项可以通过使用标准 Elasticsearch 时间单位指定，默认为 20m。

（2）cache.max.users：指定缓存用户信息的最大数量，默认为 100 000。

（3）cache.hash.algo：指定用于缓存用户信息的散列算法。

8.3.4 基于角色的访问权限配置

Security 插件会为每一个用户（包括匿名用户）分配默认的用户角色，用户角色决定了用户可以在程序中执行的操作。Security 插件内置 15 种用户角色，分别为 superuser、kibana_user、kibana_dashboard_only_user、kibana_system、logstash_admin、logstash_

system、machine_learning_admin、machine_learning_user、monitoring_user、remote_monitoring_agent、reporting_user、ingest_user、transport_client、watcher_admin 和 watcher_user。这些用户角色的权限是固定的，不能被更改。这些用户角色的权限如下：

- superuser：拥有对整个集群中所有索引和数据的完全控制权限，此外还可以管理用户和角色，该角色用户可以扮演任何其他用户角色。出于该角色的特殊性，将该角色分配给用户时要格外谨慎。
- kibana_user：拥有访问 Kibana 的最低权限，能够访问 Kibana 中的分片并监视集群。
- kibana_dashboard_only_user：拥有对 Dashboard 界面只读的权限，该角色用户不能编辑界面中的内容。
- kibana_system：拥有访问 Kibana 的各项读写权限以及对 Elasticsearch 集群的监控权限。
- logstash_admin：拥有对 Logstash 存储在 Elasticsearch 中的索引的各项管理权限。
- logstash_system：拥有向 Elasticsearch 传输系统级数据的权限，该角色用户不能访问 Logstash 索引中的内容。
- machine_learning_admin：拥有对 Machine Learning 组件相关索引的只读权限。
- machine_learning_user：拥有对 Machine Learning 组件的最小权限，包括查看该组件的配置、状态和结果等数据的权限。
- monitoring_user：拥有访问 Monitoring 的最小权限而非 Kibana 的访问权限，能够监视集群中的索引。该角色的用户应同时被分配 kibana_user 角色。
- remote_monitoring_agent：拥有从远程 Monitoring 程序向当前集群写入数据的最小权限。
- reporting_user：拥有访问 Reporting 程序的特定权限而非 Kibana 的访问权限，能够访问 Reporting 的分片。该角色用户应同时被分配 kibana_role 角色和访问生成报告所需数据的角色。
- ingest_user：拥有对所有 index 和 pipeline 配置的管理权限，但不能创建分片。
- transport_client：拥有从 Java Transport Client 段访问集群的权限，能够获取集群节点中的信息。该用户角色在使用 Transport Client 时分配。
- watcher_admin：拥有向 Watcher 组件相关索引写入数据、读取索引历史数据以及运行监视器操作的权限。
- watcher_user：拥有对 Watcher 组件相关索引的只读权限以及查看监视器操作和状态的权限。

在 Kibana 的 Management 界面中，单击页面上方的 Roles 标签即可查看系统中的用户

角色列表。单击任意一个用户角色可以转到详细信息页面，包括用户角色的名称、在集群中的特权、指定的用户和指定的分片等信息。

在 8.3.1 节中，已经添加过一个名为 cy 的 transport_client 权限用户（图 8.6），然而用户 cy 无权对 Monitoring 进行访问，因为 cy 没有被授予 monitoring_user 和 kibana_user 角色的权限。下面就以授权访问 Monitoring 为例进行说明。登录超级管理员账户 elastic，在 Management 程序中添加一个新的用户 monitor，为其分配 monitoring_user 和 kibana_user 两种用户角色的权限，使其有权访问 Monitoring 程序。

在用户列表页面右上方单击 + Create user 按钮，在添加用户界面中，为新用户起一个名字（这里为 monitor），可以设置用户的密码、全名和电子邮件地址。在下方的用户角色选择部分，为该用户指定 monitoring_user 和 kibana_user，如图 8.7 所示。单击 Save 按钮保存即可。退出登录，使用账号 monitor 登录到 Kibana，单击界面左侧 Monitoring 程序的导航按钮，会发现该用户可以进入 Monitoring 程序进行操作。单击未授权给该用户的其他导航按钮，则界面会显示出拒绝访问的信息。

图 8.7 向新用户授予 Monitoring 程序访问权限

在为用户分配角色的基础上，也可以创建新的用户角色，以便将特定的权限授予用户。这里仍以授权访问 Monitoring 为例，创建一个单独的用户身份。

首先登录超级管理员账户 elastic，创建一个新的用户角色，在角色列表页面右上方单击 + Create role 按钮，可以添加一个新的自定义用户角色，例如，设定用户名为 monitor_admin，其拥有 monitor 集群特权，下方指定了该用户角色能够访问的索引以及对数据执行的操作方式，如图 8.8 所示。其中，索引指定了 .monitoring-es-* 、.monitoring-* 和 .kibana* ，对数据的访问特权包括 manage、read、index 和 delete，实际上这些正是 monitoring_user 和 kibana_user 两种用户角色权限的并集。单击 Save 按钮保存该用户角色。

转到用户管理列表页面，单击用户 monitor 的用户名，将其用户角色改为自定义的 monitor_admin，单击 Save 按钮保存用户设置。退出当前账号，使用 monitor 账号登录后，单击界面左侧的导航按钮 Monitoring 即可再次访问 Monitoring，这一次的用户权限与上文中设置的相同。

图 8.8 添加拥有 Monitoring 程序访问权限的独立用户角色

8.3.5 IP 地址过滤

Security 插件的另一项安全功能是限制访问的 IP 地址(包括节点或 TransportClient 以及正在连接到集群中的节点等),能够被过滤的还有主机、域名和子网等。当某个节点的 IP 地址被加入黑名单中,该节点到 Elasticsearch 的连接仍会被允许,但连接后立刻就会被切断,其发出的请求也不会被执行。IP 地址过滤功能可以在 elasticsearch.yml 配置文件中添加允许访问和拒绝访问的 IP 地址,这两种设置项分别为 xpack.security.transport.filter.allow 和 xpack.security.transport.filter.deny,配置完成后重启 Elasticsearch 即可。

> Tips Elasticsearch 并不是为公网的访问而设计的,Security 插件中拥有 IP 地址过滤和其他安全机制,不代表其能抵御来自公网的安全风险。Elastic 公司建议用户不要将程序对公网开放。

代码段 8.5 实现了对特定的两个 IP 地址的过滤。

代码段 8.5: 对特定 IP 地址进行过滤

```
xpack.security.transport.filter.allow: "192.168.0.1"
xpack.security.transport.filter.deny: "192.168.0.0/24"
```

在配置中可以使用关键字_all来指出明确指定的IP之外的其余所有IP地址。代码段8.6实现了允许一组IP地址访问,并拒绝其他所有IP地址访问的配置。

代码段 8.6: 允许一组 IP地址访问,拒绝其余 IP地址访问

```
xpack.security.transport.filter.allow: [ "192.168.0.1", "192.168.0.2", "192.168.0.3", "
192.168.0.4" ]
xpack.security.transport.filter.deny: _all
```

配置中还支持IPv6的地址格式,如代码段8.7所示。

代码段 8.7: 对 IPv6地址进行过滤

```
xpack.security.transport.filter.allow: "2001:0db8:1234::/48"
xpack.security.transport.filter.deny: "1234:0db8:85a3:0000:0000:8a2e:0370:
7334"
```

Tips IPv6地址长度4倍于IPv4地址长度,表达的复杂程度也高于IPv4地址。IPv6地址的基本表达方式是X:X:X:X:X:X:X:X,其中X是一个4位十六进制整数,每个地址包括8个整数。某些IPv6地址中可能包含一长串的0,此时允许省略这一长串的0。例如,地址2000:0:0:0:0:0:0:1可以表示为2000::1,0:0:0:0:0:0:10.0.0.1可以表示为::10.0.0.1。::等同于IPv4的0.0.0.0(全0),而::1则等同于127.0.0.1(本机地址)。

当DNS功能可用时,可以指定主机名来进行IP地址过滤,如代码段8.8所示。

代码段 8.8: 对主机名进行 IP地址过滤

```
xpack.security.transport.filter.allow: localhost
xpack.security.transport.filter.deny: '*.google.com'
```

在某些情况下,禁用IP地址过滤功能可以小幅提升系统性能。禁用IP地址过滤也是在elasticsearch.yml配置文件中配置的。例如,如果要禁用Transport协议的IP地址过滤而启用HTTP的IP地址过滤,那么按照代码段8.9所示的配置信息完成配置即可。

代码段 8.9: 启用或禁用 IP地址过滤

```
xpack.security.transport.filter.enabled: false
xpack.security.http.filter.enabled: true
```

当运行在云主机这样高动态 IP 地址的环境时，进行以上配置时可能很难了解具体要过滤的 IP 地址。这时可以使用 Cluster Update API 对正在运行的集群更新配置，而不必修改配置文件并重启节点。代码段 8.10 使用 curl 实现了对 IP 地址过滤配置的动态更新。

代码段 8.10：动态更新 IP 地址过滤配置

```
curl -H 'Content-Type: application/json' -XPUT localhost:9200/_cluster/
settings -d '{
    "persistent" : {
        "xpack.security.transport.filter.allow" : "172.16.0.0/24"
    }
}'
```

类似地，对 IP 地址过滤功能的禁用也可以动态执行，如代码段 8.11 所示。

代码段 8.11：动态禁用 IP 地址过滤

```
curl -H 'Content-Type: application/json' -XPUT localhost:9200/_cluster/
settings -d '{
    "persistent" : {
        "xpack.security.transport.filter.enabled" : false
    }
}'
```

8.3.6 带有身份认证的 TransportClient

在 Java API 方面，Security 插件含有对 TransportClient 类的安全性支持，能创建与 Elasticsearch 的安全连接。要实现这样的安全连接，需要具备两个前提条件：

（1）拥有一个带有 transport_client 用户角色的 Kibana 用户。

（2）在开发平台的库中添加 X-Pack 插件的全部依赖 JAR 包（位于 Elasticsearch 安装主目录下的 plugins/x-pack 目录中）、TransportClient 的依赖 JAR 包和 XPackClient 的依赖 JAR 包。

在用户角色列表中单击 transport_client，在其详细信息中可以发现，该用户角色对索引和访问特权的设置都是空的，这将导致分配了该角色的用户在执行查询时没有足够的访问权限。如果 Java 程序在用户认证信息中指定的权限不足，在执行时将会报错。要拥有足以执行查询角色的用户，可以登录超级管理员账号 elastic，在用户角色列表中添加一个新的用户角色，这里设定用户名为 transport_admin，在集群特权中勾选 monitor 和 manage，在界面下方的索引中指定 *（代表任何索引），右侧的特权指定 all（代表全部特权）。设置完成后单

击 Save 按钮保存用户角色。转到用户列表中，将用户的用户角色 transport_client 移除，添加刚创建的 transport_admin 即可。

对于 XPackClient 依赖，可以像 4.1.1 节使用 TransportClient 时使用 Maven 导入依赖那样，在 pom.xml 配置文件中添加配置，如代码段 8.12 所示。

代码段 8.12：使用 Maven 获取 XPackClient 依赖

```
<project …>
  <repositories>
    <!-- 添加 Elasticsearch 库 -->
    <repository>
      <id>elasticsearch-releases</id>
      <url>https://artifacts.elastic.co/maven</url>
                                        <!-- 指定镜像的网络地址为 Elastic 官网 -->
      <releases>
        <enabled>true</enabled>
      </releases>
      <snapshots>
        <enabled>false</enabled>
      </snapshots>
    </repository>
    …
  </repositories>
  …
  <dependencies>
    <!-- 添加 X-Pack 依赖 JAR 包 -->
    <dependency>
      <groupId>org.elasticsearch.client</groupId>   <!-- pom.xml 中要添加的配置信息 -->
      <artifactId>x-pack-transport</artifactId>           <!-- 依赖 JAR 包名称 -->
      <version>6.2.4</version>                    <!-- 此处的版本应与 X-Pack 一致 -->
    </dependency>
    …
  </dependencies>
  …
</project>
```

除此之外，也可以通过访问 Elastic 官网直接获取依赖 JAR 包，然后导入开发平台的 library 中。获取依赖 JAR 包资源链接为 https://artifacts.elastic.co/maven/org/elasticsearch/client/x-pack-transport/6.2.4/x-pack-transport-6.2.4.jar（其中，6.2.4 是 X-Pack 对应的版本号，读者应根据实际情况修改这两个版本号）即可直接获取依赖 JAR 包。

带有 X-Pack 的 TransportClient 程序的写法与第 4 章(如代码段 4.2)介绍的方法略有不同。首先,初始化 Client 的类由 PreBuiltTransportClient 变为 PreBuiltXPackTransportClient。其次,初始化过程中传入的 Settings 类型参数不再为空,而是指定了身份认证的相关信息。代码段 8.13 实现了经过身份认证的 TransportClient 访问 Elasticsearch 索引文件并查询文档数据的功能。其中,cluster.name 后面的参数为集群的名称;xpack.security.user 后面的参数为 Security 插件中指定的具有访问 TransportClient 权限的用户名和密码,书写格式为"用户名:密码"。

代码段 8.13: 使用带有身份认证的 TransportClient 查询文档数据

```
import org.elasticsearch.xpack.client.PreBuiltXPackTransportClient;
//其他 import 语句,略
public class ClientTest {
    private static final Logger logger= (Logger) LogManager.getLogger(ClientTest.
    class);
    public static void main(String[] args) throws IOException, ExecutionException,
    InterruptedException{
      TransportClient client= new PreBuiltXPackTransportClient(Settings.builder()
        .put("cluster.name", "cyElasticsearch")          //指定集群名称
        .put("xpack.security.user", "cy:123456")      //指定拥有访问权限的用户
        .build())
        .addTransportAddress(
            new InetSocketTransportAddress(InetAddress.getByName("localhost"),
            9300));
     QueryBuilder qb= matchAllQuery();
     SearchResponse response= client.prepareSearch("it-home")
        .setTypes("_doc")
        .setSearchType(SearchType.DFS_QUERY_THEN_FETCH)
        .setQuery(qb)
        .setFrom(0)
        .setSize(10)
        .setExplain(true)
        .get();
     SearchHit[] hits= response.getHits().getHits();
     for(SearchHit hit : hits) {
        System.out.println(hit.getSourceAsString());
     }
     client.close();
   }
}
```

Tips 在上面代码中，认证信息是以明文形式传输的。如果要进行更高加密级别的传输，可以使用 SSL 加密等其他形式的配置，详见 Elastic 官网有关内容的介绍。

8.3.7 带有身份认证的 RESTful 命令

由于 Security 插件安全性的作用，之前在终端中使用的 curl 命令不再直接执行，没有身份认证信息的命令将直接被拒绝。要使用 curl 执行带有身份认证的命令，需要在命令中加入-u 参数或--user 参数（注意，后面这个参数以两个横线开头），后面跟“用户名：密码”格式的身份认证信息。代码段 8.14 演示了添加新索引文件 idx 的命令。

代码段 8.14：使用带有身份认证的 curl 命令

```
curl -H 'Content-Type: application/json' --user cy:123456 -XPUT 'localhost:9200/idx'
```

8.4 使用 Monitoring 监控系统运行状态

X-Pack 中的 Monitoring 组件能够完成在 Kibana 中对 Elasticsearch 执行监控的任务。通过 Monitoring 可以查看集群健康度和实时性能，并对集群、索引和节点的各项指标进行分析。此外，Kibana 自身性能也可以通过 Monitoring 来监控。从 X-Pack 被安装到集群之时起，Monitoring 即运行在每一个节点上并开始在 Elasticsearch 中收集数据。在 Kibana 中安装 X-Pack 之后，监控数据可以通过一组定制仪表板来查看。本节对 Monitoring 插件的使用和配置进行介绍。

8.4.1 系统运行状态监控

使用拥有 Monitoring 访问权限的用户（如超级管理员用户 elastic 或 8.3 节中创建的 monitor 等）登录 Kibana，单击左侧导航栏中的 Monitoring 导航按钮访问 Monitoring 界面。Monitoring 的初始界面包含两个仪表板，分别用于显示 Elasticsearch 和 Kibana 的各项运行指标，如图 8.9 所示。

在这两个仪表板上方均有对应的状态标志。在 Elasticsearch 节点数不足以分配所有分片时，其状态会显示为黄色的圆点；当集群中所有分片分配完毕时，其状态即变为绿色的圆点。

在 Elasticsearch 面板中单击 Overview 链接，可以查看 Elasticsearch 中的全部性能指标，如图 8.10 所示。界面上方列出了集群中的节点数、索引数、占用内存大小、总分片数量、

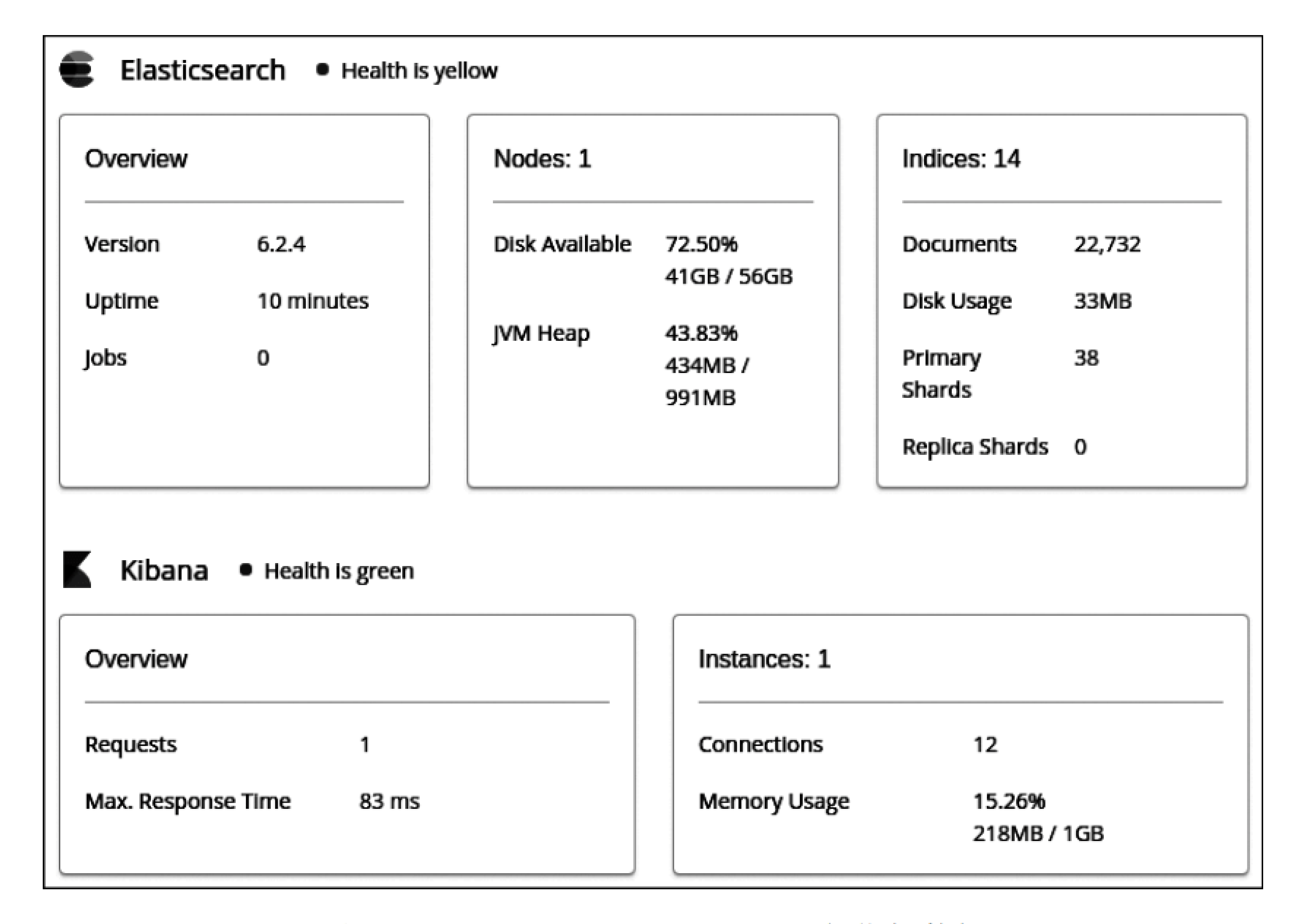


图 8.9　Elasticsearch 和 Kibana 运行指标数据

未分配分片数、存储文档数量、数据存储大小、更新次数和版本号等。界面中部绘制了 4 种折线统计图，分别为系统中的搜索和索引速率（每秒）、搜索和索引延迟（每毫秒）的实时统计。

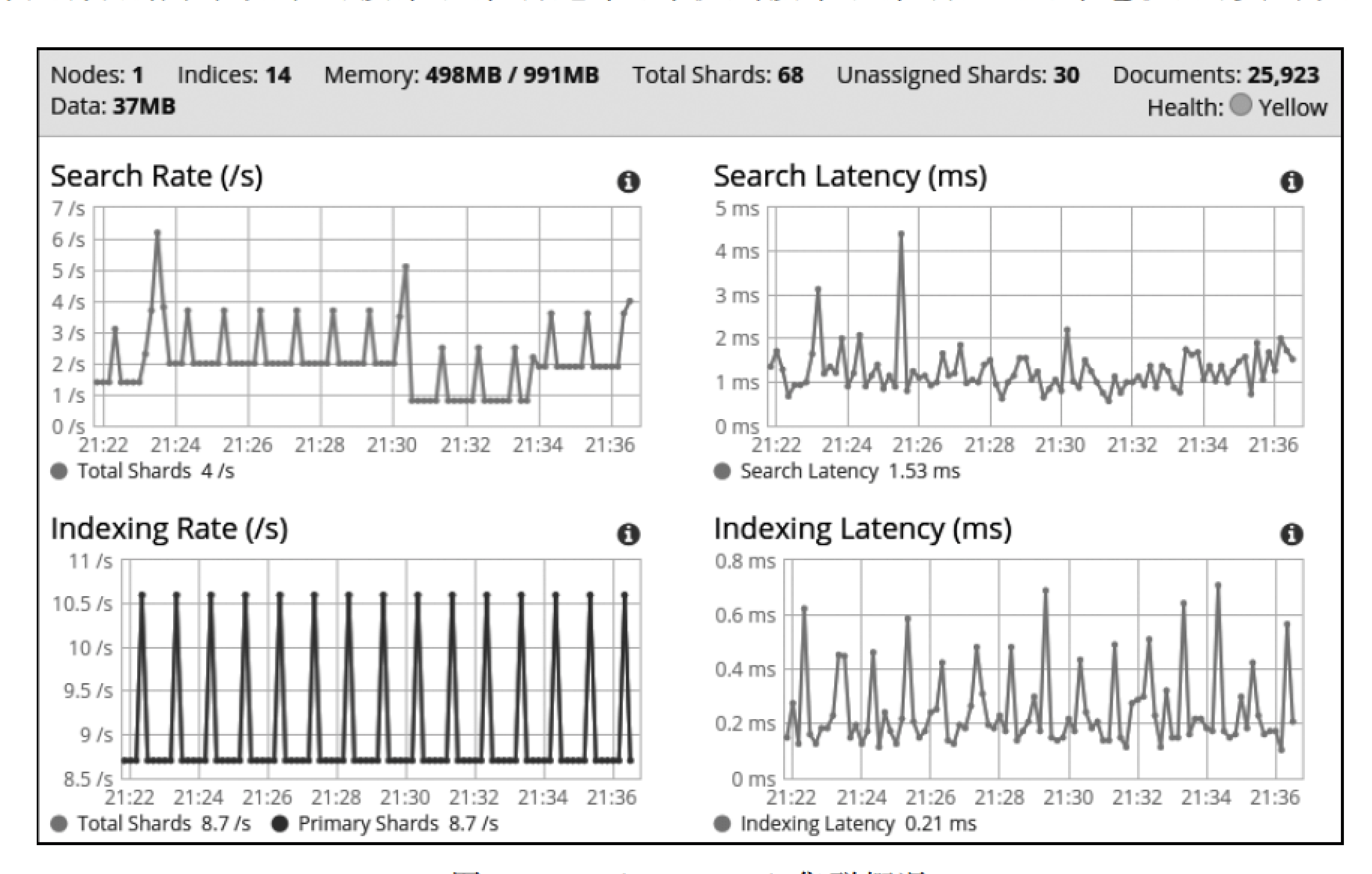


图 8.10　Elasticsearch 集群概况

单击界面上方的 Indices 标签，可以转到所有已存储索引文件的列表页面，列表中还包

含了每个索引文件中存储的文档数、数据存储量大小、索引速率、搜索速率和未分配分片数等信息，如图 8.11 所示。

Name	Status ↓	Document Count	Data	Index Rate	Search Rate	Unassigned Shards
ifeng	Yellow	942	4.7 MB	0 /s	0 /s	5
information	Yellow	5	23.2 KB	0 /s	0 /s	5
it-home	Yellow	800	5.8 MB	0 /s	0 /s	5
myweibo3	Yellow	3	13.8 KB	0 /s	0 /s	5
weibo	Yellow	3	15.4 KB	0 /s	0 /s	5
weibo2	Yellow	0	1.3 KB	0 /s	0 /s	5

图 8.11　索引文件列表

单击任意一个索引文件，可以查看该索引文件在一定时间范围内的 6 项指标，如图 8.12 所示。

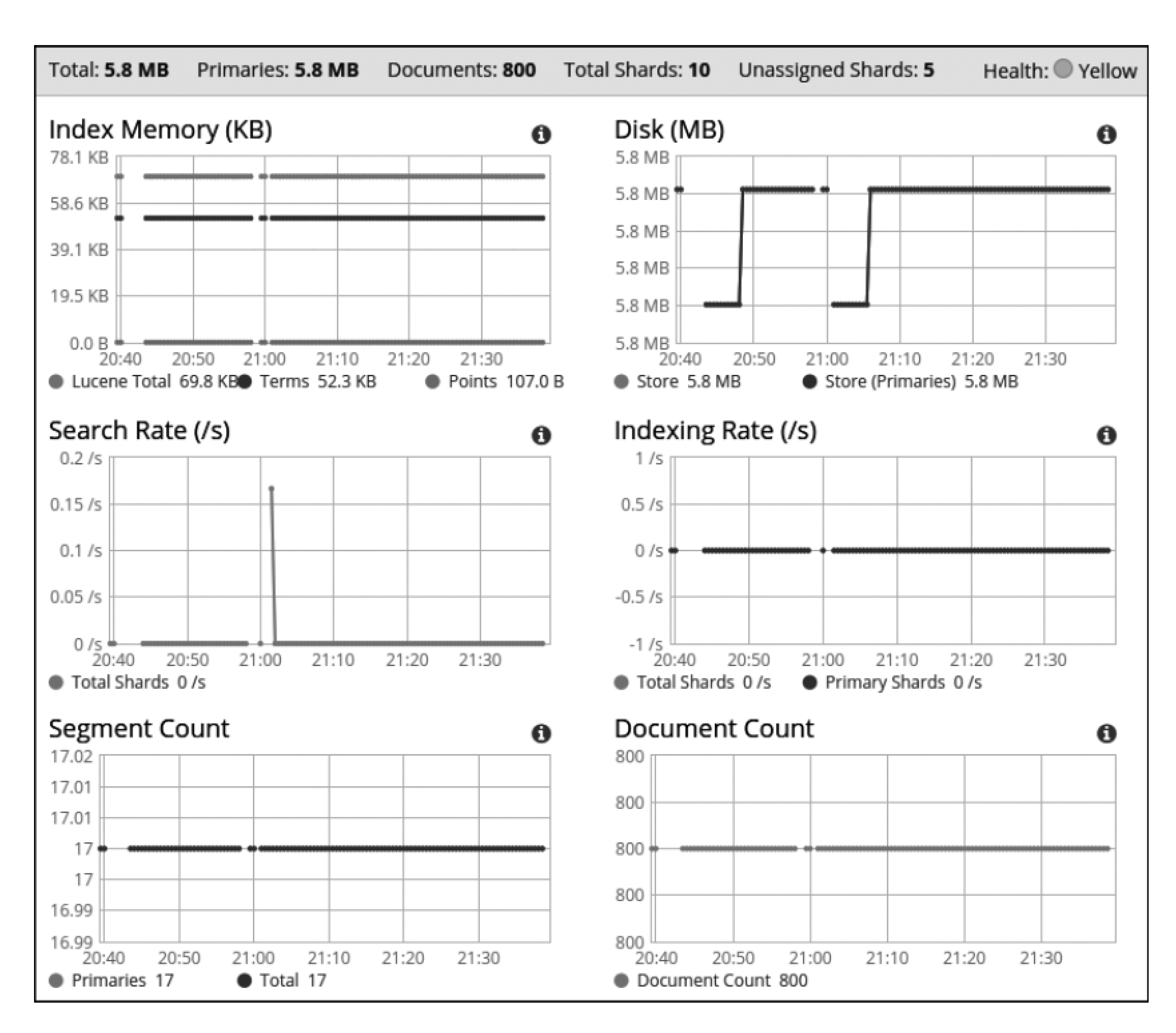


图 8.12　索引文件中的 6 项指标

返回 Elasticsearch 运行状态监控页面，在界面上方单击 Nodes 标签，可以查看已开启的节点占用系统资源的情况与趋势，相应指标有 CPU 使用率、平均负载、Java 虚拟机内存、磁盘可用空间和分片分配数量等，如图 8.13 所示。图 8.13 中左侧的 Master 为节点名称(它是之前在 elasticsearch.yml 中配置好的)，节点名称下方显示系统分配的端口地址。在中间每一项数据旁边都标有其最大值和最小值。

Name ↑	Status	CPU Usage	Load Average	JVM Memory	Disk Free Space	Shards
★ Master 127.0.0.1:9300	● Online	1 % ↓ 8 % max 0 % min	0.45 ↑ 1.46 max 0.21 min	49 % ↑ 50 % max 24 % min	40.7 GB ↓ 40.7 GB max 40.7 GB min	38

图 8.13　索引文件的指标

单击任意一个节点的名称，可以查看该节点的 6 项指标的实时统计数据，包括延迟(ms)、Java 虚拟机堆大小(GB)、索引内存(MB)、CPU 使用率、系统负载和平均分片数量等。同时，界面上方多了当前节点所占端口和节点类型(即主节点或从节点)两项信息，如图 8.14 所示。

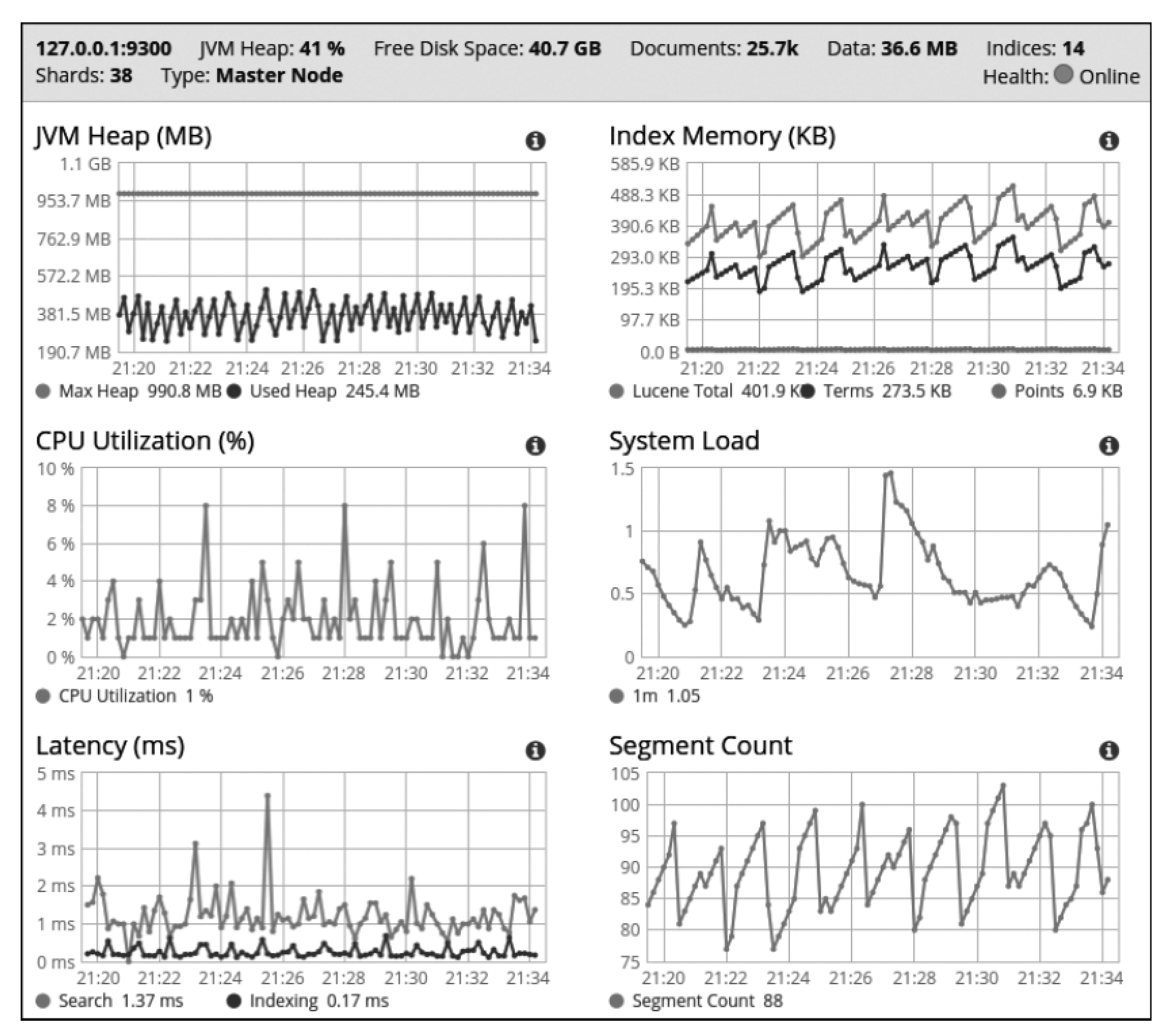


图 8.14　节点的 6 项指标

与 Elasticsearch 的监控数据相似，Kibana 同样以折线统计图的形式来监控运行状态。返回 Monitoring 程序主界面，在 Kibana 面板中单击 Overview 链接，即可查看 Kibana 的各项性能指标，包括运行实例数量、收到请求数、链接数、最大响应时间以及内存占用量等。界面中部显示两种折线统计图，分别为客户端请求数量和客户端响应时间，如图 8.15 所示。

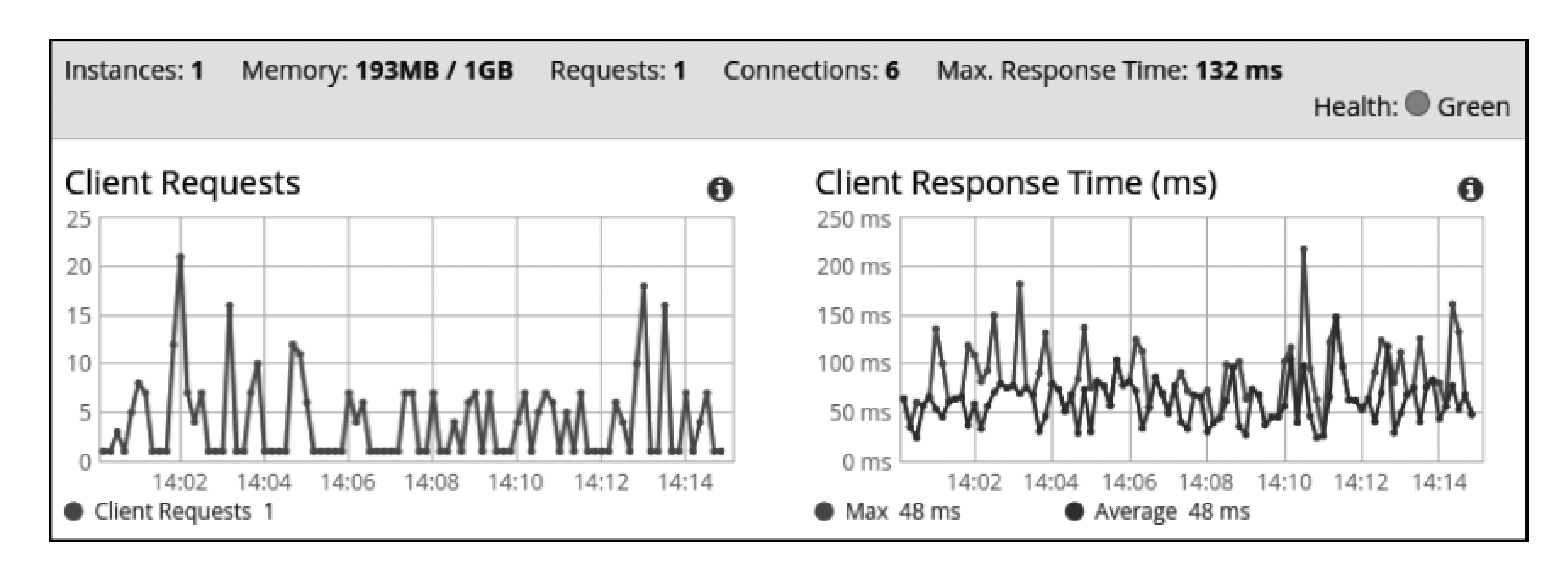


图 8.15　Kibana 运行概况

在界面上方单击 Instances 链接，可以查看 Kibana 运行实例的有关数据，包括运行状态、内存占用大小、平均负载、请求数、平均响应时间和最长响应时间等，如图 8.16 所示。

Name ↑	Status	Load Average	Memory Size	Requests	Response Times
cy-N53SN localhost:5601	Green	0.39	195.00 MB	6	37 ms avg 56 ms max

图 8.16　Kibana 运行实例数据

单击左侧名称（如此例中的 cy-N53SN），可以查看该实例相关的 6 项指标，分别为客户端请求、客户端响应时间（ms）、HTTP 连接数、占用内存大小（GB）、系统负载和事件循环延迟（ms），如图 8.17 所示。界面上方还显示了当前实例所占的端口、操作系统可分配内存大小和当前程序版本。

8.4.2　配置 Monitoring

在默认使用条件下，Monitoring 程序将每隔 10s 从全部索引文件中收集数据并存储在本地。实际使用过程中，可以对 Monitoring 程序进行配置，指定要监控的特定索引文件，向其他独立的集群导出数据，以及配置专门存储数据的索引文件等。其他配置还包括可以指定程序多长时间收集一次数据、配置超时时间限制、设置本地存储数据的保留期限等。这些配置均需要通过在 elasticsearch.yml 中添加配置信息来完成。下面对部分配置进行说明。

- xpack.monitoring.collection.cluster.state.timeout：指定收集集群状态信息的超时时长，默认为 10min。

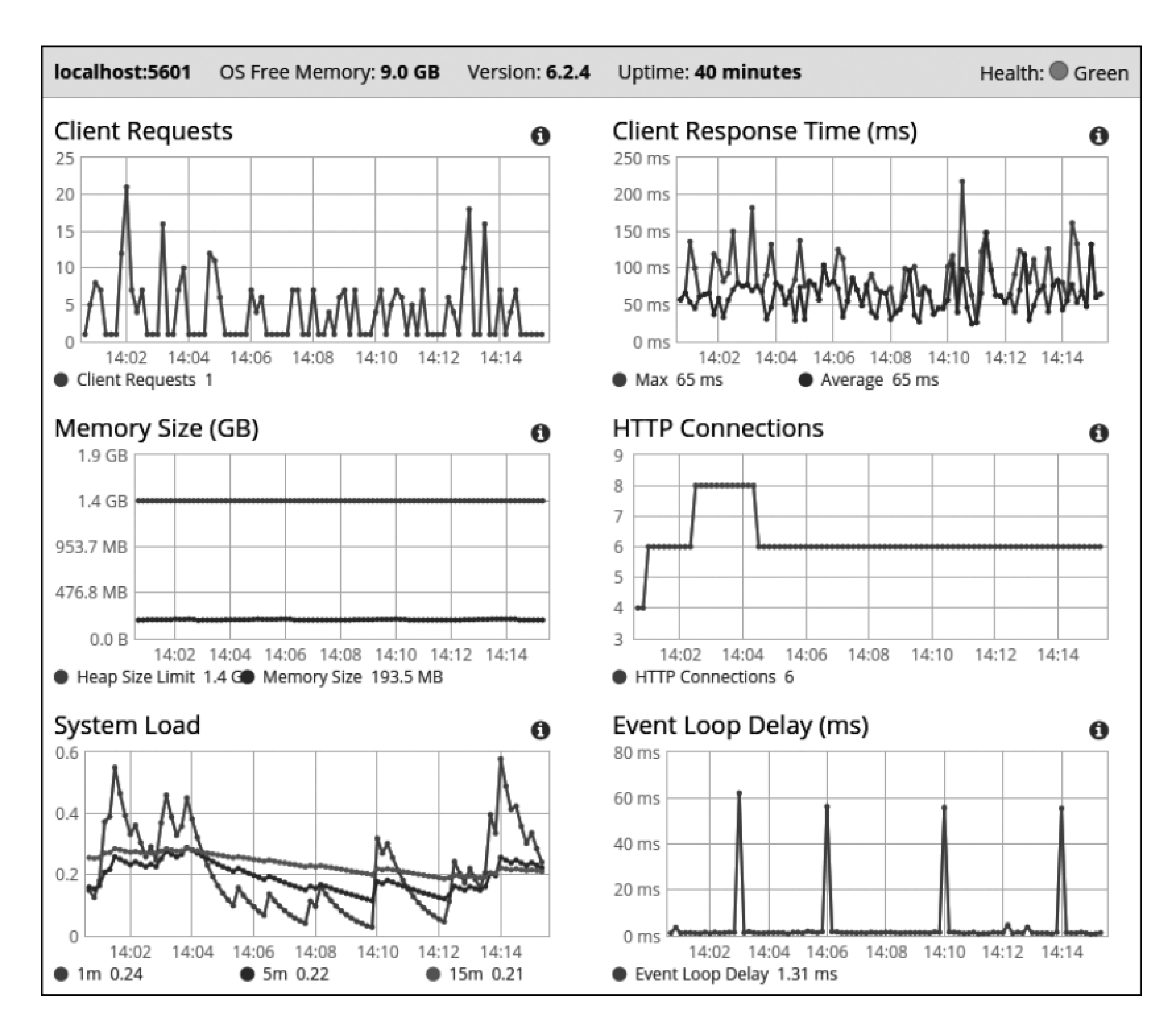


图 8.17　Kibana 运行实例 6 项指标

- xpack.monitoring.collection.cluster.stats.timeout：指定收集集群统计信息的超时时长，默认为 10min。
- xpack.monitoring.collection.indices：指定程序收集的数据存储在哪些索引文件中，默认为全部索引文件。指定索引文件时需要将不同索引文件名以逗号隔开（如“test1，test2，test3”），可以在书写索引文件名时使用通配符 *（如 logstash-*），也可以使用＋号或－号来保留或排除索引文件（如＋test*、－test3）。
- xpack.monitoring.collection.index.stats.timeout：指定收集索引文件统计数据的超时时长，默认为 10min。
- xpack.monitoring.collection.indices.stats.timeout：指定收集全部索引文件统计数据的超时时长，默认为 10min。
- xpack.monitoring.collection.interval：指定程序收集数据样本的时间间隔，默认为 10s。如果修改了该设置项，则需要同时在 kibana.yml 中添加配置项 xpack.monitoring.min_interval_seconds，并将该项的值设置为与 elasticsearch.yml 中相同的值（设为－1 表示临时禁用收集数据的功能）。

- xpack. monitoring. history. duration：设置程序保存数据的索引文件的保留期限，超过期限的索引文件将被自动删除，默认为 7d。该设置项被限定最小值为 1d，以保证监控的有效性。该项不能被禁用。

8.4.3 搭建独立的 Monitoring 集群

X-Pack 支持远程 Monitoring 端连接和存储数据。要在独立的 Monitoring 集群中存储数据，首先需要在独立的集群中安装相同版本的 Elasticsearch 和 Kibana，并为集群中的每个节点安装对应版本的 X-Pack。接下来，在 Monitoring 集群中创建一个 remote_monitor_user 角色的用户（例如设定用户名为 remote_monitor），并保证 Kibana 实例连接到 Monitoring 集群上。

Monitoring 程序使用 HTTP Exporter 来将数据指标传输到 Monitoring 集群。这样的配置需要在提供数据的集群中的 elasticsearch. yml 配置文件中添加相应的配置信息（如代码段 8. 15 所示）。添加配置后，需要重启 Elasticsearch。

代码段 8.15：配置提供数据集群的 HTTP Exporter

```
xpack.monitoring.exporters:
  id1:                                         #定义远程 Exporter 名称
    type: http                                 #指定传输方式为 HTTP
    host: ["http://es-mon-1:9200", "http://es-mon2:9200"]     #指定主机列表
    auth.username: remote_monitor              #指定具有该权限的用户
    auth.password: remote                      #指定用户的密码
```

接下来要为 Monitoring 集群的 Kibana 配置集群的连接信息。首先在提供数据的集群中创建一个拥有 Kibana 访问权限的用户（即同时拥有 kibana_user 和 monitoring_user 两种角色的用户，例如设定用户名为 kibana_monitor）。然后在 Monitoring 集群的 kibana. yml 配置文件中添加相应的配置信息（如代码段 8. 16 所示）。添加配置后，重新启动 Kibana。

代码段 8.16：为 Kibana 配置集群的连接信息

```
xpack.monitoring.elasticsearch.url: "http://es-mon-1:9200"          #指定 Monitoring 集群的地址
xpack.monitoring.elasticsearch.username: "kibana-monitor"  #指定用户名
xpack.monitoring.elasticsearch.password: "kibana"          #指定密码
```

可以在提供数据的集群中的 elasticsearch. yml 配置文件中添加如下配置信息来启用数据传输功能，配置示例如代码段 8. 17 所示。

代码段 8.17：启用 Exporter 的数据传输功能示例

```
xpack.monitoring.exporters:
  my_local:                          #定义本地 Exporter
    type: local
  my_remote:                         #定义远程 Exporter
    type: http
    host: [ "10.1.2.3", … ]
    auth:
      username: my_username
      password: my_password
    connection:                      #连接相关参数
      timeout: 6s
      read_timeout: 60s
    ssl: …                           #TLS/SSL 相关配置参数,略
    proxy:
      base_path: /some/base/path     #为发出的请求添加路径以协同 proxy 工作,可选
    headers:                         #以键-值对的形式添加请求头
      My-Proxy-Header: abc123
      My-Other-Thing: [ def456, … ]
    index.name.time_format: YYYY-MM  #设置时间格式作为后缀
```

8.5　Alerting 插件与异常事件警报

在集群的实际使用过程中，可能会遇到数据的变动或运行中的异常现象。这就需要对诸如磁盘使用情况等进行监控。例如，如果接下来的几天里，某些节点的磁盘剩余空间即将耗尽，那么系统会发出警告；或者在监控 Elasticsearch 的过程中，如果某些节点与集群断开连接，或查询吞吐量超出预期范围，那么系统应向管理员发出通知。使用 Alerting 插件可完成对上述类似情况的监视。X-Pack 推出了用来创建、管理和测试监视器的 API，监视器可被触发并执行多个预警通知操作。监视器由 schedual、query、condition 和 actions 模块构成。

- schedual：用于执行查询和判断条件的调度程序。
- query：作为 condition 的输入而运行的查询。监视器支持完整的 Elasticsearch 查询语言与聚合。
- condition：用于判断是否运行 actions 的条件，这里可以使用简单查询条件，也可以使用脚本来构建更为复杂的查询条件。
- actions：一个或多个操作，例如发送邮件、利用 webhook 软件向第三方系统推送数据或索引文件查询结果等。

监视器会在 Elasticsearch 专门的索引文件中记录下完整的历史记录，其中包含每次监

视器触发的时间、查询结果的记录、是否符合条件以及程序执行了何种操作等信息。本节对 Alerting 插件通过 RESTful 和 Java 程序两种方式的配置方法进行介绍。

8.5.1 通过 RESTful 方式设置监视器

在通过 RESTful 方式设置检索器时，对监视器的设置包括对 trigger、input、condition、transforms 和 actions 5 个部分的配置。其中，trigger 是每个监视器中都有的设置项，仅有 schedule trigger 一种，决定了监视器被触发的时间，例如每隔 10s 触发一次；input 为监视器载入监视系统有效负载提供数据，分为 simple、search、http 和 chain 4 种，如对 Elasticsearch 索引的检索等；condition 对监视器的操作是否执行进行控制，分为 always、never、compare、array compare 和 script 5 种，如果不指定 condition，那么默认设置为 always；transforms 是一个可选项，为监视器操作的执行准备数据结果，分为 search、script 和 chain 3 种；actions 是指定当符合条件时执行何种操作，分为 email、webhook、index、logging、hipchat、slack、pagerduty 和 jira 8 种。

以检测系统中的 404 错误信息并报告给第三方集群和管理员为例，代码段 8.18 演示了通过 Kibana 中的 Dev Tools 向系统中添加监视器来检测系统错误信息的方法。

代码段 8.18：添加监视器以检测系统的错误信息

```
PUT _xpack/watcher/watch/log_errors
{
  "metadata" : {                              #为监视器附加静态的元数据,可选
    "color" : "red"
  },
  "trigger" : {
    "schedule" : {
      "interval" : "10s"                      #指定监视器每 10s 触发一次
    }
  },
  "input" : {
    "search" : {                              #input 使用 search 方式
      "request" : {
        "indices" : "whale",                  #指定索引文件
        "body" : {
          "size" : 0,
          "query" : { "match" : { "status_code" : "404" } }
                                                   #指定在 status_code 字段中检索错误信息
        }
      }
    }
  },
  "condition" : {                             #condition 使用 compare 方式
```

```
    "compare" : { "ctx.payload.hits.total" : { "gt" : 5 }}
                                          #当检索出 5条以上错误时执行操作
  },
  "transform" : {                         #符合条件时为执行操作准备数据,可选
    "search" : {                          #transform使用 search方式
      "request" : {
        "indices" : "whale",              #指定索引文件
        "body" : {
          "query" : { "match" : { "status_code" : "404" } }
                                          #在 status_code字段中检索错误信息
        }
      }
    }
  },
  "actions" : {
    "my_webhook" : {                      #将错误信息报告给第三方集群
      "webhook" : {                       #此处使用 webhook方式
        "method" : "POST",
        "host" : "第三方主机名",
        "port" : 9200,
        "path" : "/{{watch_id}}",
        "body" : "Encountered {{ctx.payload.hits.total}} errors"
      }
    },
    "email_administrator" : {             #将错误信息以电子邮件形式报告管理员
      "email" : {                         #此处使用 email方式
        "to" : "<email>"                  #将<email>替换为管理员电子邮件地址
        "subject" : "Encountered {{ctx.payload.hits.total}} errors",
        "body" : "Too many error in the system, see attached data",
        "attachments" : {
          "attached_data" : {
            "data" : {
              "format" : "json"
            }
          }
        },
        "priority" : "high"
      }
    }
  }
}
```

在上面的代码中,要实现发送邮件的功能,需要在 elasticsearch.yml 配置文件中配置收发邮件双方的电子邮件账户信息,如代码段 8.19 所示。程序运行后,管理员邮箱中将可能

收到来自 Alerting 插件的电子邮件，如图 8.18 所示。

代码段 8.19：在 elasticsearch.yml 中配置电子邮件账户信息

```
xpack.notification.email.account:
  work:
    profile: qq
    email_defaults:
      from:<email>            #将<email>替换为管理员电子邮件地址
    smtp:
      auth: true
      starttls.enable: true
      host: smtp.qq.com
      port: 587
      user:<username>         #将<username>替换为自己的邮件用户名
      password:<password>     #将<password>替换为自己的邮件密码
```

主题	时间↓	大小
404 recently encountered	36秒前	1.3K
Encountered 6 errors - Too many error in the system, see attached data	1分钟前	5.1K

图 8.18　来自 Alerting 插件的电子邮件

通过代码段 8.20 中的命令，可以查看检测到的相关错误的历史记录。

代码段 8.20：查看检测到的相关错误的历史记录

```
GET .watcher-history*/_search?pretty
{
  "query" : {
    "bool" : {
      "must" : [
        { "match" : { "result.condition.met" : true }},
        { "range" : { "result.execution_time" : { "to" : "now" }}}
      ]
    }
  }
}
```

要检索程序中的监视器，可以执行代码段 8.21 所示的命令，该命令检索程序中前 10 个监视器的信息。

代码段 8.21：检索前 10 个监视器的信息

```
GET .watches/_search
{
  "size" : 100
}
```

要删除一个监视器，可以执行代码段 8.22 所示的命令。

代码段 8.22：删除监视器

```
DELETE _xpack/watcher/watch/log_errors          #log_errors是监视器的名字
```

8.5.2 通过 Java 程序设置监视器

X-Pack 插件中提供了名为 WatcherClient 的 Java 类库，添加了对本地监视器的支持。8.3.6 节已经提到过关于获取 XPackClient 依赖的方法，在此基础上使用 Put Watch API 编写 Java 代码实现对监视器的设置，使用 schedule trigger 每分钟触发一次监视器，使用 search input 来获取状态码为 404、出现在最近 5min 内的信息，使用 condition 判定是否有文档数据被检索到，一旦发现符合条件的数据，action 则发送电子邮件给管理员，如代码段 8.23 所示。

代码段 8.23：编写 Java 程序来设置监视器

```
TransportClient client= new PreBuiltXPackTransportClient(Settings.builder()
    .put("cluster.name", "cyElasticsearch")
    .put("xpack.security.user", "cy:123456")
    .build())
    .addTransportAddress(new InetSocketTransportAddress(InetAddress.getByName("localhost"), 9300));
XPackClient xpackClient= new XPackClient(client);
WatcherClient watcherClient= xpackClient.watcher();
WatchSourceBuilder watchSourceBuilder= WatchSourceBuilders.watchBuilder();
//设置触发器
watchSourceBuilder.trigger(TriggerBuilders.schedule(Schedules.cron("0 0/1 * * ?")));
//为 input 创建检索
SearchRequest request= Requests.searchRequest("whale").source(searchSource()
    .query(boolQuery()
```

```
        .must(matchQuery("status_code","404"))
        .filter(rangeQuery("timestamp").gt("{{ctx.trigger.scheduled_time}}"))
        .filter(rangeQuery("timestamp").lt("{{ctx.execution_time}}"))
        ));
    //创建检索输入
    SearchInput input= new SearchInput(new WatcherSearchTemplateRequest(
        new String[]{"whale"}, null, SearchType.DEFAULT,
        WatcherSearchTemplateRequest.DEFAULT_INDICES_OPTIONS,
        new BytesArray(request.source().toString())), null, null, null);
    //设置输入
    watchSourceBuilder.input(input);
    //设置触发条件
    watchSourceBuilder.condition(new ScriptCondition(new Script("ctx.payload.hits.total> 1")));
    //为发送电子邮件的操作创建邮件模板
    EmailTemplate.Builder emailBuilder= EmailTemplate.builder();
    emailBuilder.to("<email>");      //将<email>替换为具体的电子邮件地址
    emailBuilder.subject("404 recently encountered");
    EmailAction.Builder emailActionBuilder= EmailAction.builder(emailBuilder.build());
    //创建 action
    watchSourceBuilder.addAction("email_someone", emailActionBuilder);
    PutWatchResponse putWatchResponse= watcherClient.preparePutWatch("my- watch")
        .setSource(watchSourceBuilder)
        .get();
    //程序最后关闭客户端
    client.close();
```

编写代码时，使用静态导入的 builder 可以简化和压缩代码。PutWatchResponse 类的使用如代码段 8.24 所示。

代码段 8.24：简化的 PutWatchResponse 类使用

```
PutWatchResponse putWatchResponse2= watcherClient.preparePutWatch("my- watch")
        .setSource(watchBuilder()
        .trigger(schedule(cron("0 0/1 * * * ?")))
        .input(searchInput(new WatcherSearchTemplateRequest(new String[]{"whale"},
```

```
        null, SearchType.DEFAULT,
        WatcherSearchTemplateRequest.DEFAULT_INDICES_OPTIONS, searchSource()
          .query(boolQuery()
            .must(matchQuery("status_code", 404))
            .filter(rangeQuery("timestamp").gt("{{ctx.trigger.scheduled_time}}"))
            .filter(rangeQuery("timestamp").lt("{{ctx.execution_time}}"))
          ).buildAsBytes())))
      .condition(compareCondition("ctx.payload.hits.total", CompareCondition.Op.GT, 1L))
      .addAction("email_someone", emailAction(EmailTemplate.builder()
        .to("<email>")          //将<email>替换为具体的电子邮件地址
        .subject("404 recently encountered"))))
      .get();
```

要检索一个监视器，可以使用 Get Watch API，如代码段 8.25 所示。

代码段 8.25：检索一个监视器

```
GetWatchResponse getWatchResponse= watcherClient.prepareGetWatch("my- watch").get();
XContentSource source= getWatchResponse.getSource();
Map<String, Object> map= source.getAsMap();
```

要删除一个监视器，可以使用 Delete Watch API，通过监视器的 id 将其删除。删除监视器后，索引文件. watches 中有关该监视器的所有文档信息将会消失，该监视器将无法再执行，但是该监视器运行的历史记录将会保留。代码段 8.26 实现了将 id 为 my-watch 的监视器删除的功能。

代码段 8.26：删除一个监视器

```
DeleteWatchResponse deleteWatchResponse= watcherClient.prepareDeleteWatch
("my- watch").get();
```

8.5.3 使用 Watcher UI 管理监视器

为方便管理监视器，X-Pack 插件提供了可视化的监视器管理界面。进入 Management 界面后，在 Elasticsearch 部分单击 Watcher 链接即可跳转至 Watcher UI 界面。该界面直观地列出了系统中所有的监视器程序，用户可以查看各种监视器的状态，如图 8.19 所示。界面中还提供了创建和删除等管理功能，用户可以通过直接与该界面交互来完成对监视器的管理。

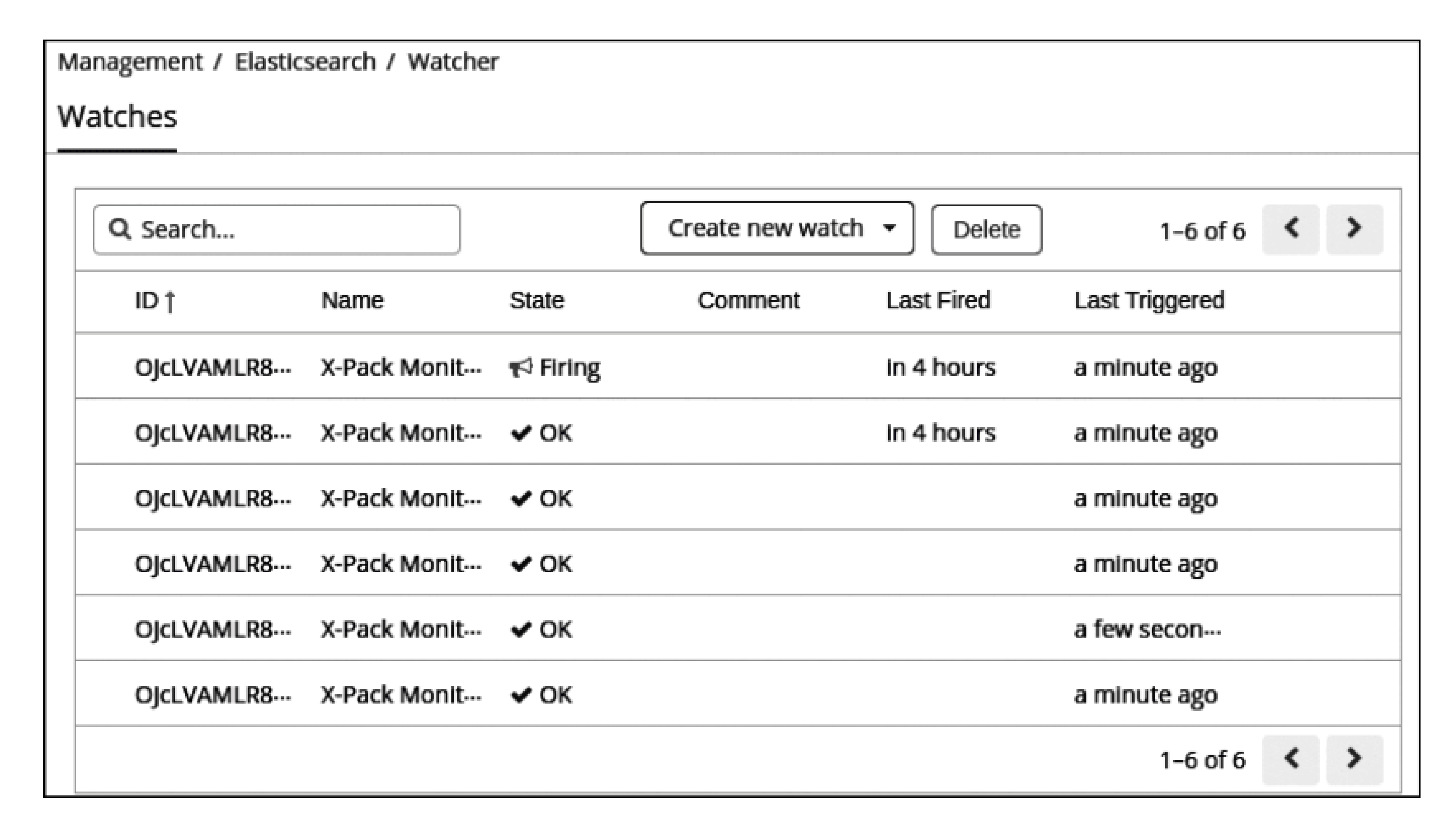


图 8.19　Watcher UI 界面

8.6 Reporting 与报告生成

X-Pack 插件为 Kibana 的可视化提供了生成报告的功能，报告中可以包含 Kibana 中的 Dashboard、Visualize 等程序可视化统计图表和已保存的检索。本节对 Reporting 插件的使用和配置等进行介绍。

8.6.1　在程序中生成报告

要使用 Reporting 手动生成报告，需要先以一个拥有 reporting_user 角色的用户身份登录到 Kibana。此时可以直接使用 elastic 这样的超级管理员用户，也可以利用 elastic 用户创建一个专门的新用户，分配 reporting_user 角色、kibana_user 角色和访问生成报告所需数据的角色。

在 Kibana 中安装 X-Pack 后，对于一些与可视化相关的程序（如 Dashboard、Visualize）和已保存的检索，在其界面上方会出现 Reporting 按钮，单击这个按钮可以进行 PDF 文档生成、分享链接生成等操作，如图 8.20 所示。

在 Reporting 面板中单击 Printable PDF 按钮，程序会将当前打开的可视化统计图表加入生成 PDF 的队列，其进度可以在 Management 程序中 Kibana 部分的 Reporting 中查看，如图 8.21 所示。列表中每一行代表一个任务。生成完毕后，右侧会出现下载按钮，单击该按钮即可获取已生成的 PDF 文档。新版 X-Pack 提供了十分便捷的操作方式，在 PDF 文档生成操作完成时，用户浏览的当前界面顶部会弹出通知条，其中提供了下载按钮。

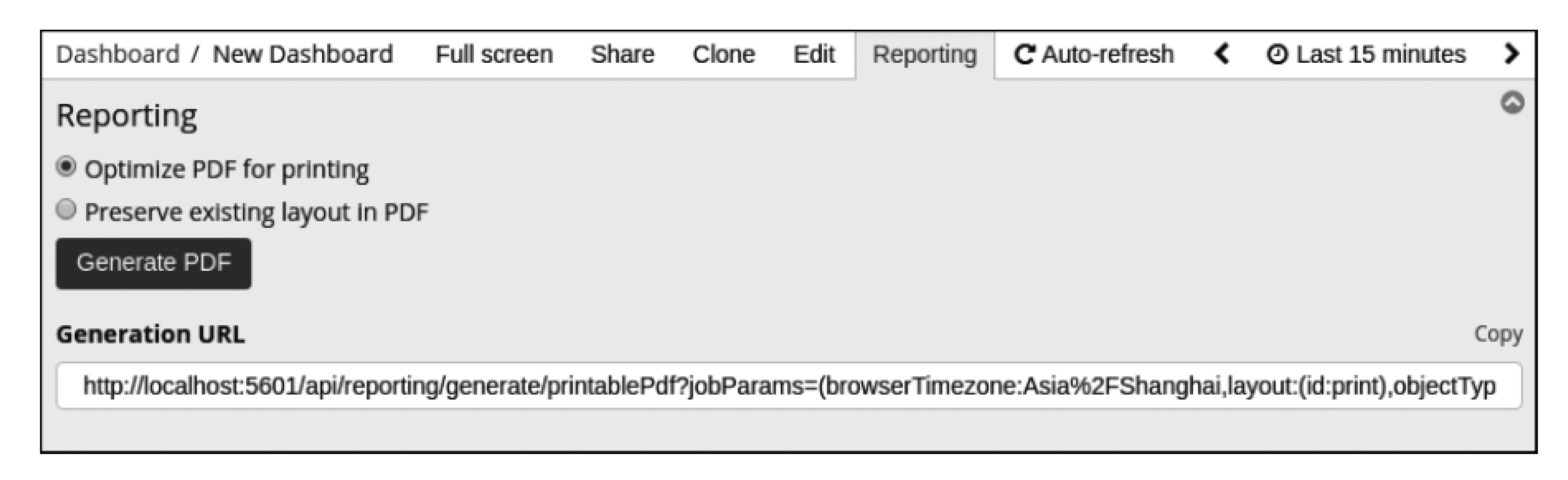


图 8.20 在界面上方展开 Reporting 面板

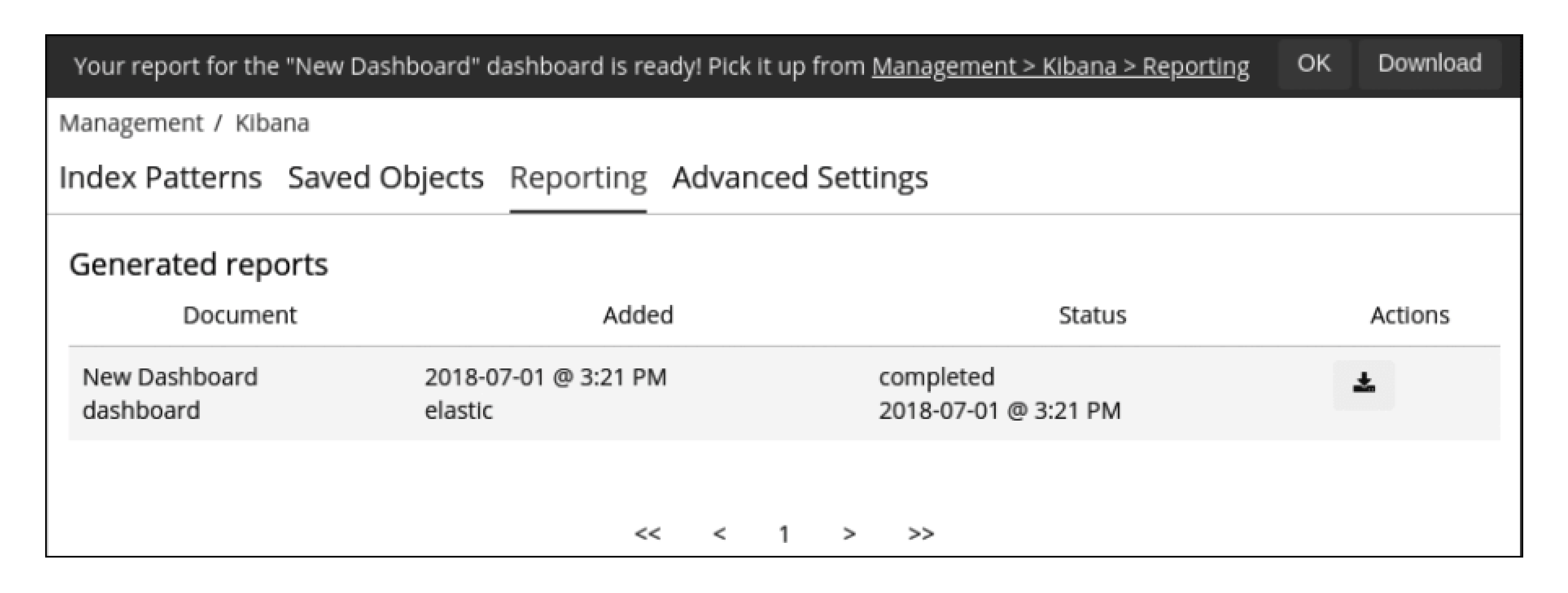


图 8.21 生成 PDF 文档

8.6.2 通过监视器自动生成报告

可以用上文中提到过的监视器来自动生成报告。在图 8.20 中，Generation URL 处的链接可以用来请求报告的内容。在以下 3 种报告中，链接的格式是不同的：

- Dashboard 程序报告中的链接格式为/api/reporting/generate/dashboard/<dashboard-id>&sync。
- Visualize 程序报告中的链接格式为/api/reporting/generate/visualization/<visualization-id>&sync。
- 已保存的检索报告中的链接格式为/api/reporting/generate/search/<saved-search-id>&sync。

其中，<×××-id>指的是各自的 object 名称。链接中使用参数_g 来指定可视化应限定的时间间隔。下面以每小时生成检索的报告为例，利用监视器来发送电子邮件给管理员，如代码段 8.27 所示。

代码段 8.27：自动生成报告，并发送电子邮件给管理员

```
PUT _xpack/watcher/watch/search_report
{
  "trigger": {
    "schedule": {
      "interval": "1h"                      #每隔 1h 触发一次
    }
  },
  "actions": {
    "email_admin": {
      "email": {
        "to": "'<username> <email>'",       #指定管理员的用户名和电子邮件地址
        "subject": "Monitoring Report",
        "attachments": {
          "search_report.pdf": {            #指定输出文件名
            "http": {
              "content_type": "application/pdf",
              "request": {
                "method": "POST",
                "headers": {
                  "kbn-xsrf": "reporting"
                },
                "read_timeout": "300s",     #设置生成超时时长为 300s
                 "url": "http://localhost:5601/api/reporting/generate/search/whale?_g=(time:
                 (from:now-1d%2Fd,mode:quick,to:now))&sync"   #Reporting生成链接
              }
            }
          }
        }
      }
    }
  }
}
```

上面的代码中使用了发送邮件的功能，此处应参照代码段 8.20 所示的配置信息，在 elasticsearch.yml 中配置好电子邮件账户。此外，请求报告的时间间隔应大于报告生成的时间间隔，必要时应适当增加请求报告的时间间隔；报告生成的时间间隔默认为 30s，如果由于某些原因，例如报告中的信息较为复杂，或生成报告的计算机运行较慢，导致常规生成时间超过 30s，那么应该在 elasticsearch.yml 配置文件中调整报告生成超时时长。

8.7　使用 Graph 探索数据关联

X-Pack 中的 Graph 能够对 Elasticsearch 索引文件中的数据进行分析，并为用户展示不同数据之间的关联，这有利于发现海量数据之间的关联。基于这些信息，用户可以在欺诈检测、信息推荐等多个应用领域对数据潜在的作用进行挖掘。本节对 Graph 插件的使用进行介绍。

在 Graph 中，一个图表示 Elasticsearch 索引中一组相关信息的网络。其中，文档数据中的每个词项为一个节点，两个词项之间的联系为一条边。Elasticsearch 为 Graph 提供了用来生成边的聚合，可用来在所有数据中寻找最有意义的关联。Graph API 利用 Elasticsearch 的相关性评分和排序工具来检索和分析数据。

在 Kibana 中，单击左侧导航栏中的 Graph 导航按钮即可进入 Graph。刚进入时，整个页面几乎是空白的，只有上方提供了选择索引模式、字段和输入查询条件的文本框等功能，如图 8.22 所示。

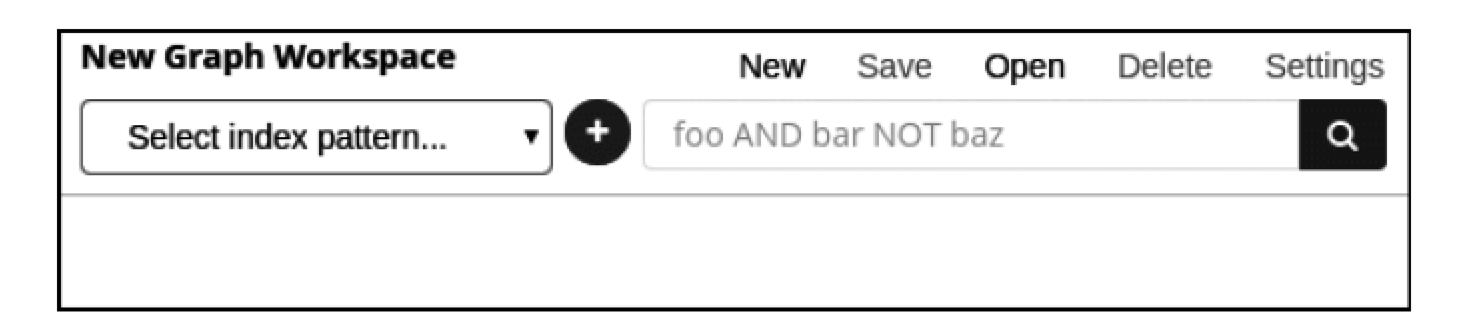


图 8.22　Graph 插件初始界面

首先，在左边的 Select index pattern 下拉列表中选择一个索引模式。如果需要某个索引模式出现在列表中，应事先在 Management 中的 index pattern 中添加，此处可选择 whale，随后右侧添加字段的⊕按钮变为可用，单击该按钮，下方会弹出字段列表，如图 8.23 所示。

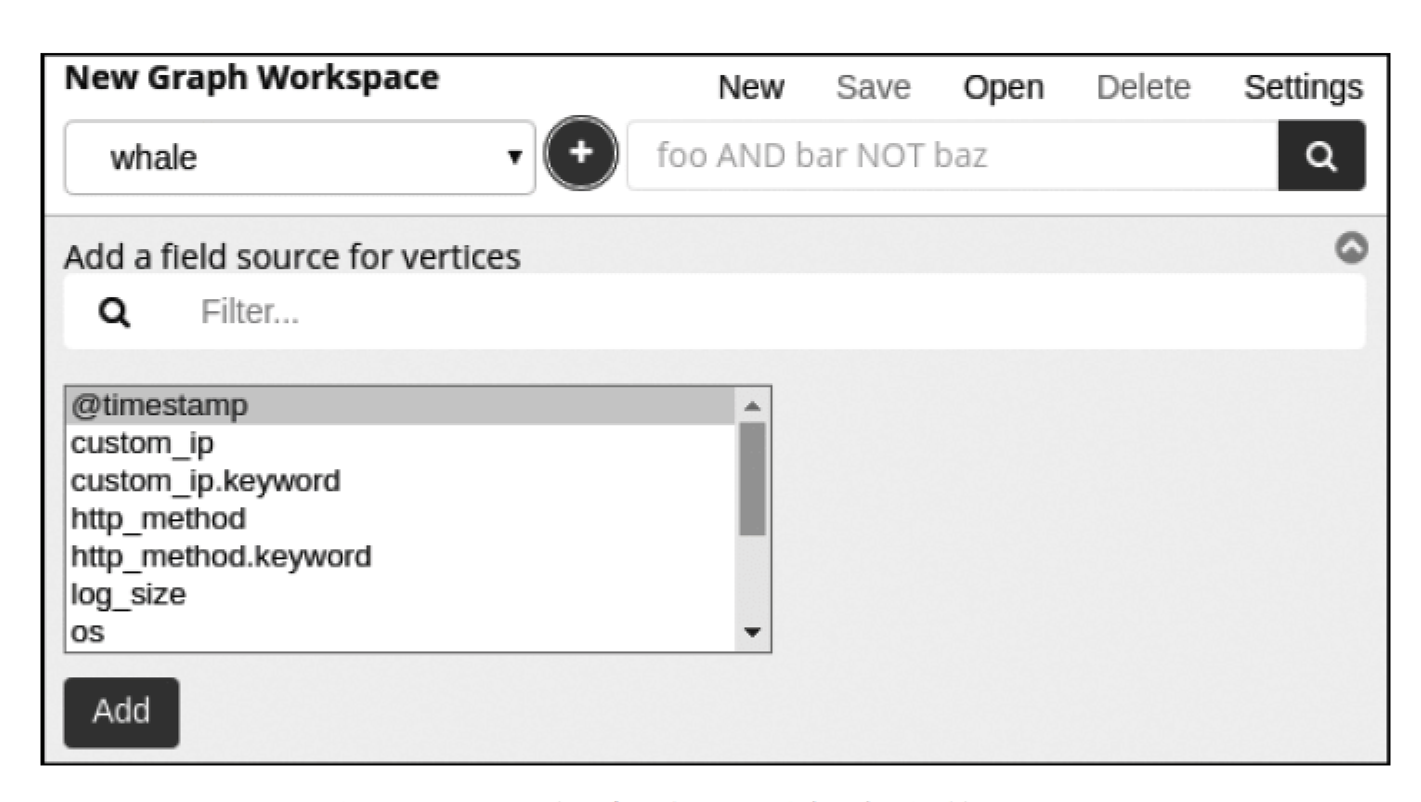


图 8.23　选择索引后，添加字段按钮可用

选择一个或多个字段，这里选择了 os、status_code 和 timestamp 3 个字段，并将每个字段的 Max terms per hop 属性设为 10。此时，后面的查询条件输入框变为可用。该输入框支持 Lucene 中的查询语法（例如 query string 等）。在其中输入 * 表示查询全部内容，如图 8.24 所示。

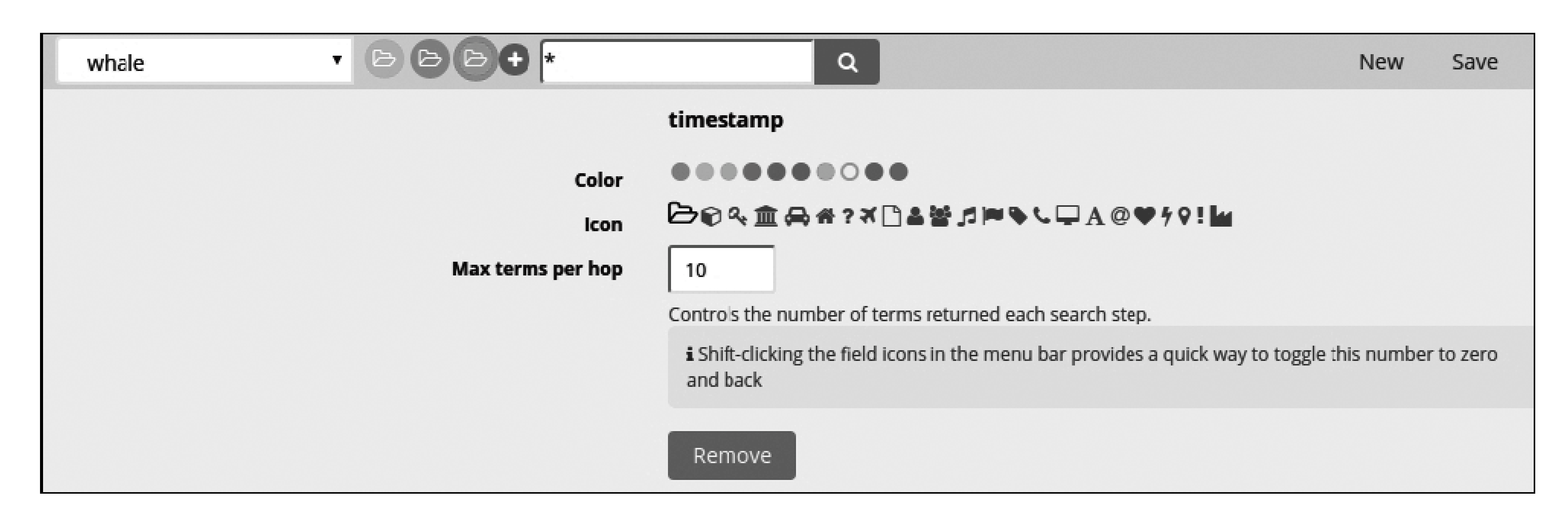


图 8.24　查询全部信息

单击右侧的查询按钮，界面中即出现如图 8.25 所示的 Graph，它是由点和边组成的图，图中每个点都是文档中的词项，图中的连线显示了词项之间的关联。

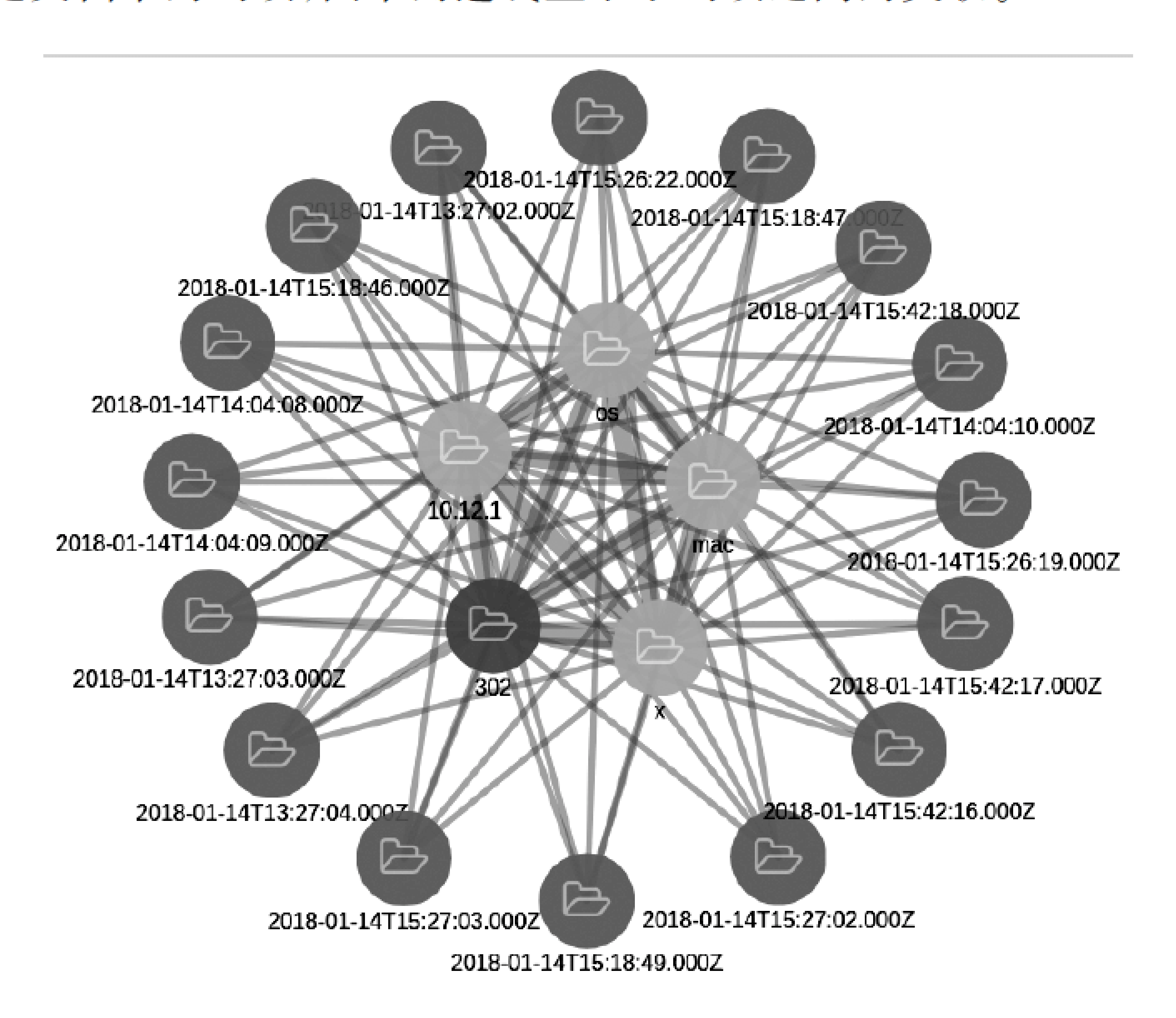


图 8.25　基于查询结果创建的 Graph

单击图中的任意一条边，可以查看两点之间的关联数据，如图 8.26 所示。其中，上部的 2018-01-14T15:27:02 为 timestamp 字段的数据，302 为 status_code 中的数据，两端的 按

钮可以实现将该点向另一侧合并；中间的韦恩图中，两个圆形的大小和相对位置描述了两点的数量和包含关系；下部左侧的数字 360 表示索引中包含词项 2018-01-14T15：27：02 的文档数，右侧的数字 8731 表示索引中包含词项 302 的文档数，中间括号中的数字 360 表示索引中同时包含两种词项的文档数。

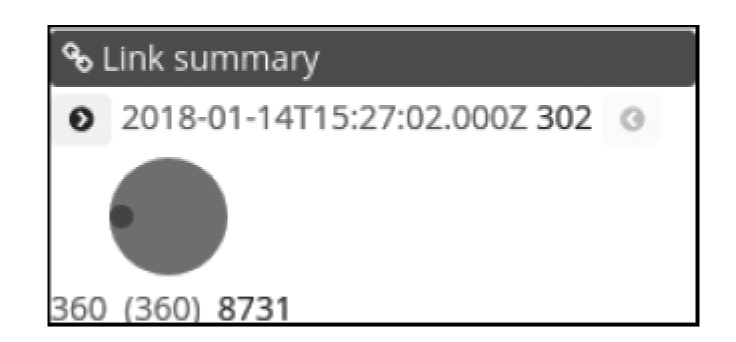


图 8.26　两点间的关联数据

8.8　使用 Machine Learning 发现数据趋势异常

随着数据量及其复杂度的日益增加，通过人工方式在仪表板中检查数据错误、网络攻击等问题变得不切实际。X-Pack 插件中的 Machine Learning 组件可自动对基于时间序列数据的正常行为进行建模，通过学习数据的趋势和周期性等信息，实时地识别数据异常，简化根本原因分析过程，并减少误报。

Machine Learning 组件是以任务(job)为基本单位执行的。在其 Job Management 选项卡中单击 Create new job 按钮即可创建一个新任务。接着，选择一个索引，进入向导选择界面。界面中提供了 4 种向导，分别为 Single metric、Multi metric、Population 和 Advanced。本节以 Single metric 为例，对任务的创建进行介绍。

单击 Single metric，界面跳转到单指标任务创建向导页面。在页面中的 Aggregation 下拉列表中选择 Sum，Field 下拉列表中选择 # views，Bucket span 设置为 7d(即 7 天的时间间隔)，如图 8.27 所示。



图 8.27　设定新任务的聚合、字段和 bucket 时间间隔

单击图 8.27 中右侧的 ▶ 按钮，界面中会出现数据随时间变化的趋势折线图，如图 8.28 所示。

在界面下方设置任务名称，添加任务描述。如果需要给任务分组，可以在分组输入框中输入组名。最后单击 Create Job 按钮，即可创建该任务。然后，在 Machine Learning 任务列表中启动该任务，并设定起止日期，程序即开始学习数据的变化趋势，并在未来对可能的异常做出判断，如图 8.29 所示。

单击图 8.29 中的 View Results 按钮，可以跳转到 Single Metric Viewer 选项卡，查看程序对

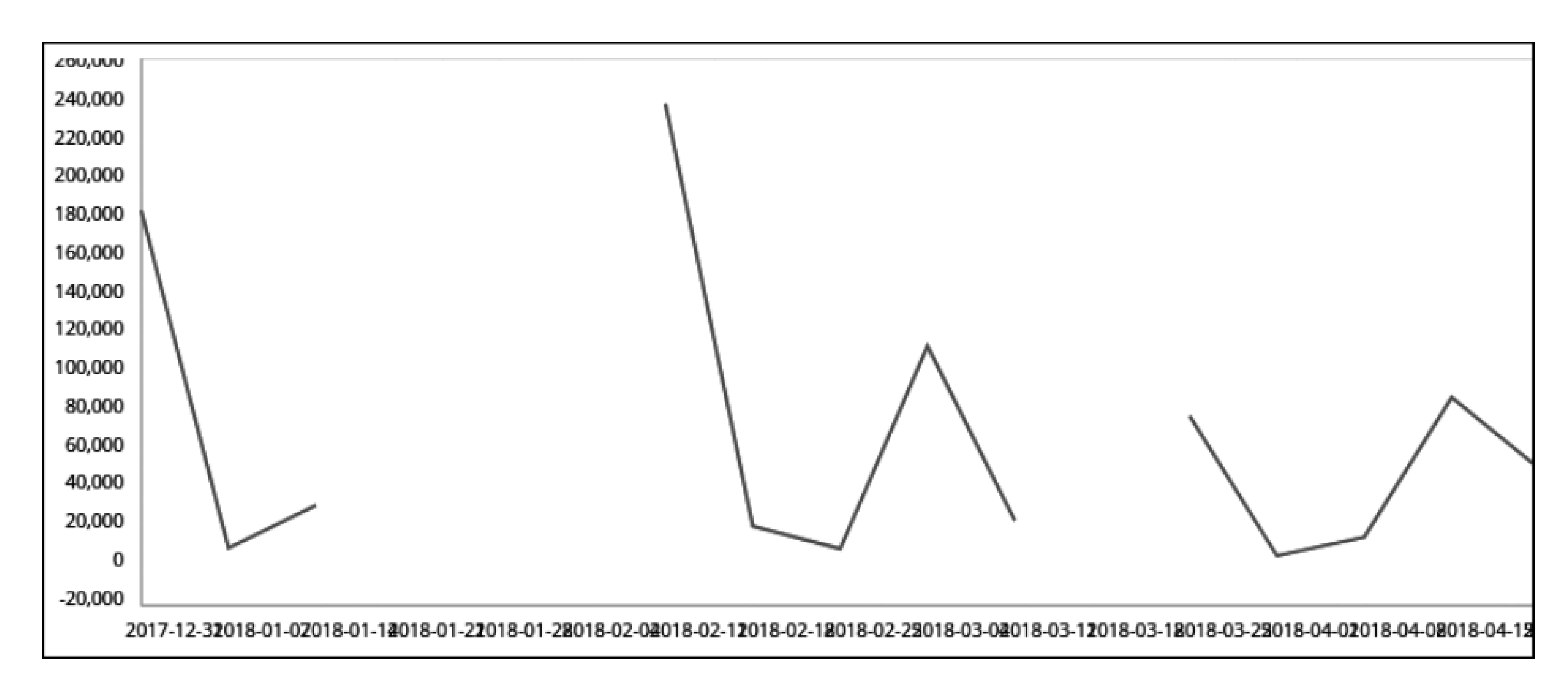


图 8.28　数据变化趋势折线图

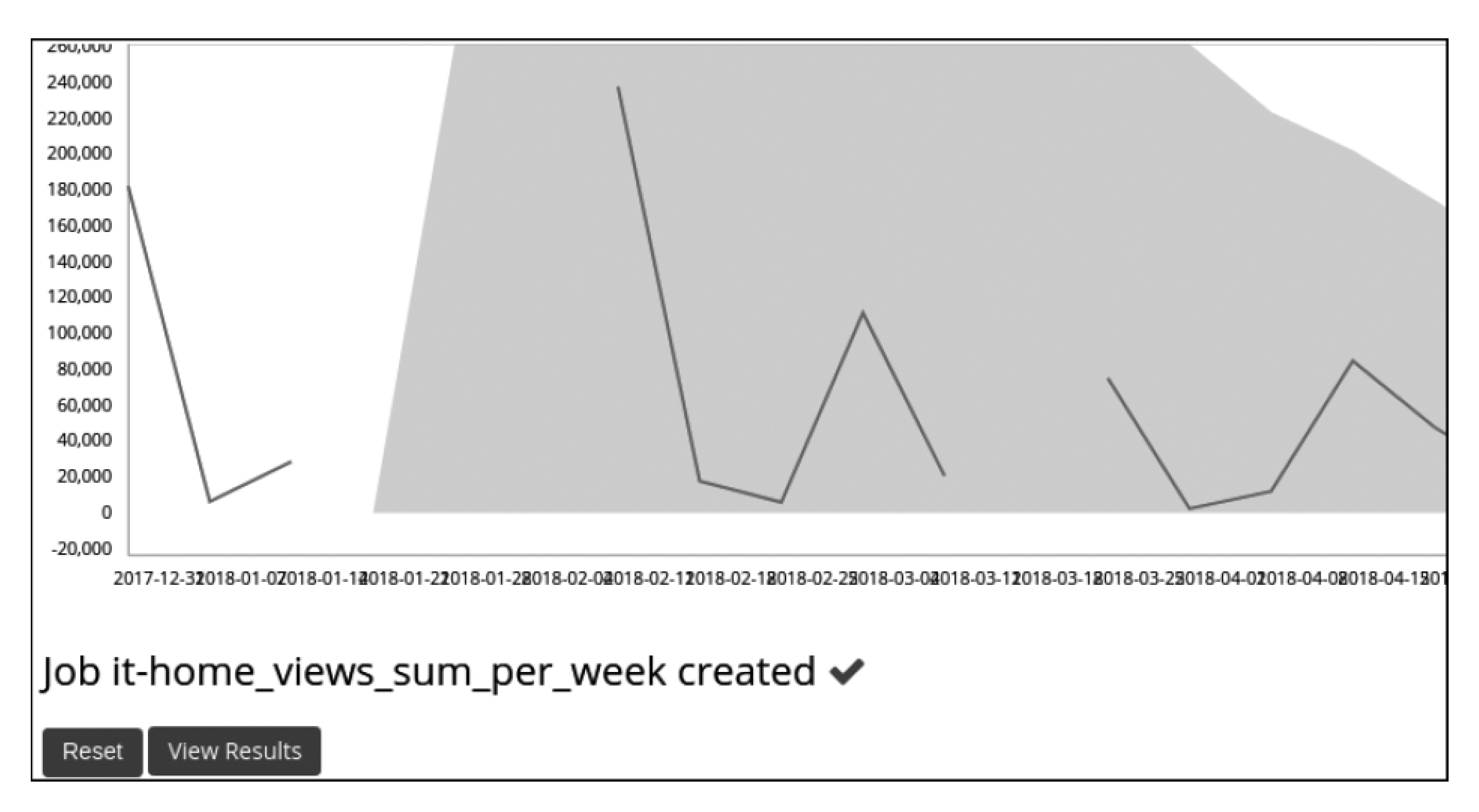


图 8.29　程序开始学习数据变化趋势

数据是否异常做出的判断，如图 8.30 所示，一旦程序发现异常，将会在时间轴上以红色标记高亮显示。

在任务的创建过程中，在选择向导的界面中选择 Data Visualizer，可以查看数据中的各种指标和字段的详细数据统计展示。图 8.31 展示了 it-home 索引文件统计信息中的两项，左侧为程序对数值型字段 views 的 metric 统计信息，其中包含该字段的最小值、平均值、最大值等统计数据展示；右侧为对非数值型字段 editorName 的 bucket 统计信息，其中包含该字段不同内容的数量统计数据展示。

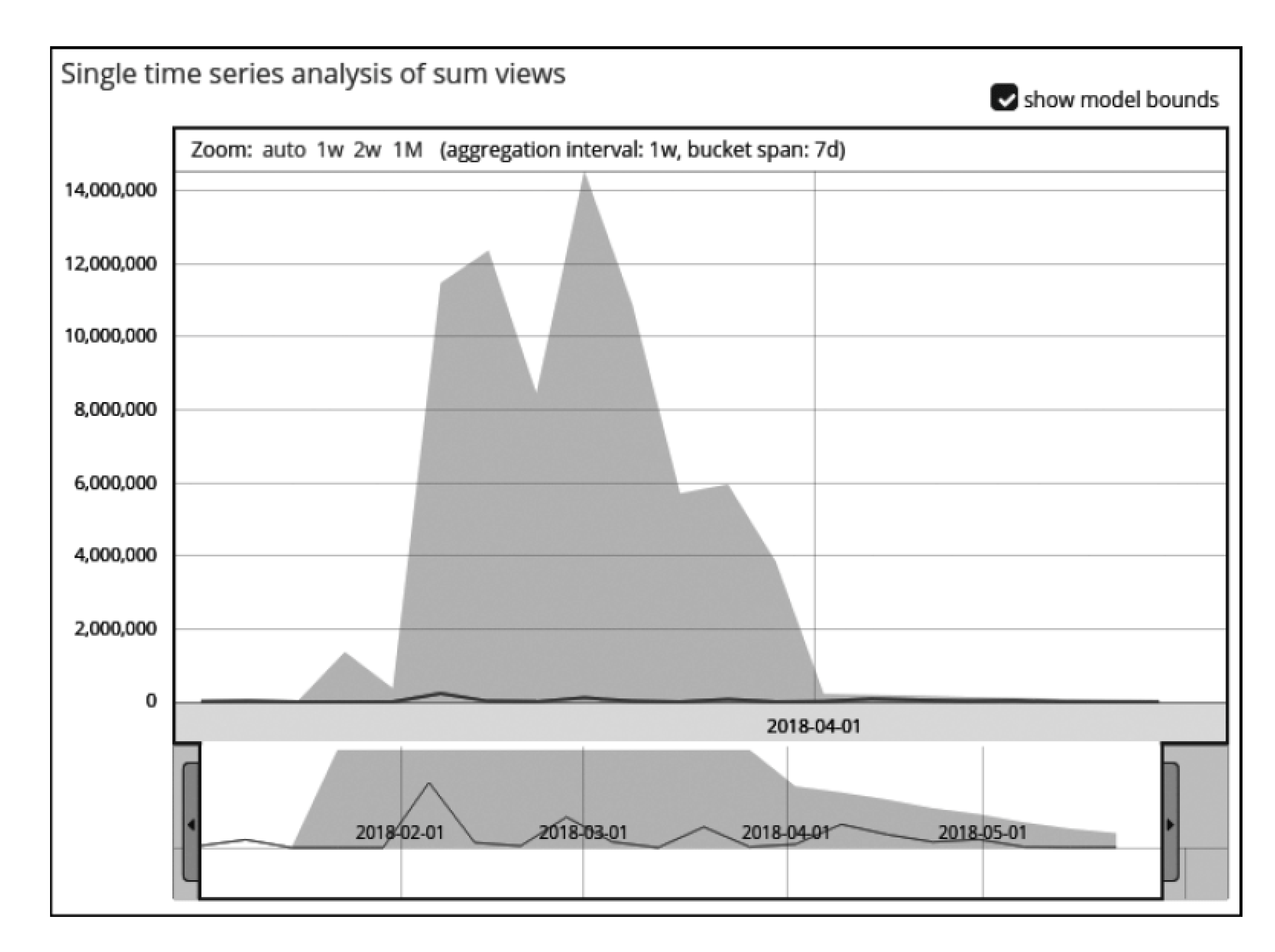


图 8.30　查看程序对数据是否异常的判断

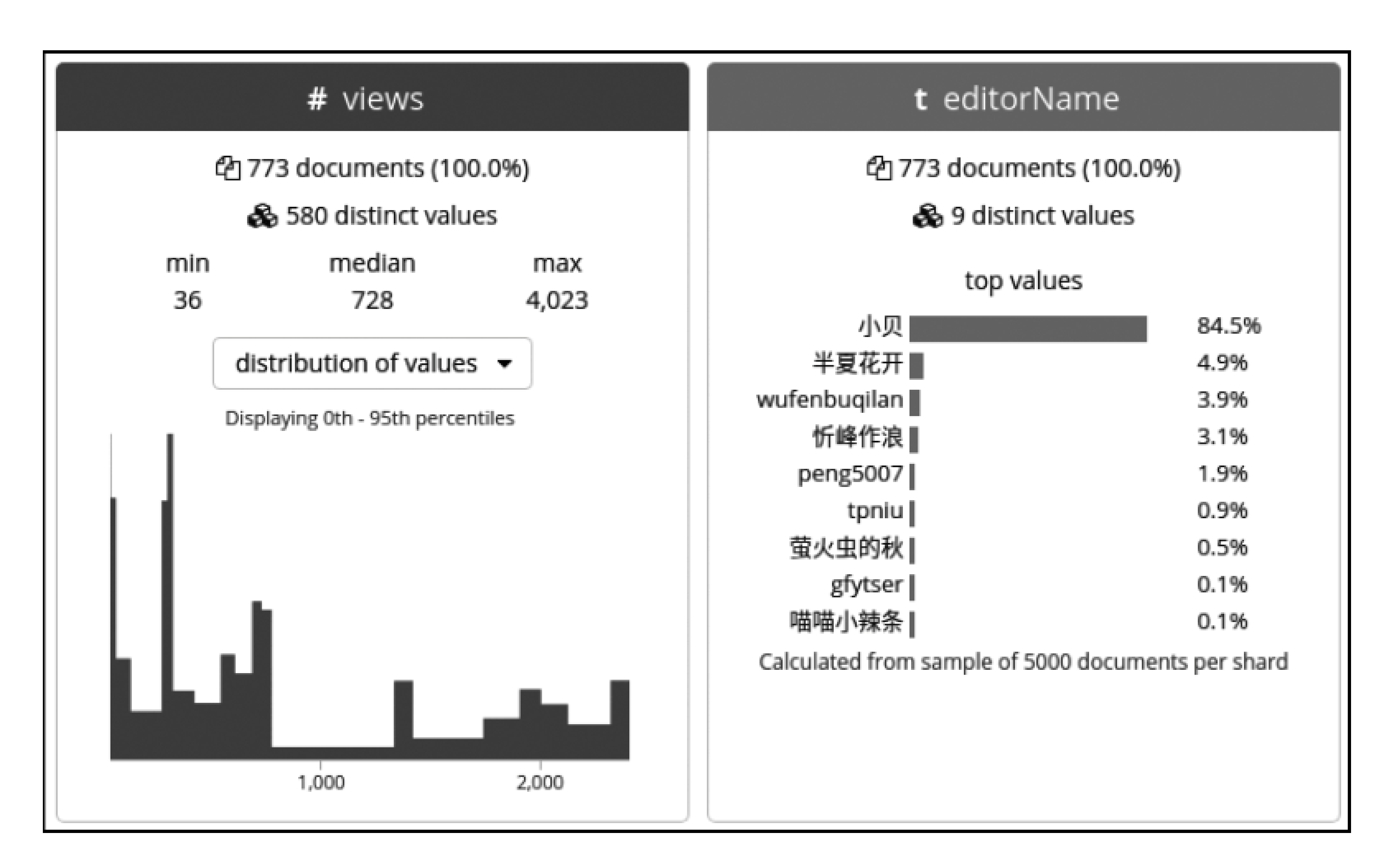


图 8.31　针对指标和字段的详细数据统计展示

8.9　使用 Search Profiler 分析搜索查询

Elasticsearch 拥有一个功能强大的分析器 API，可用于检查和分析搜索查询。但其对于搜索的回应是一个非常庞大的 JSON 数据，很难进行人工分析。为此 X-Pack 提供了

Search Profiler 工具，可以将 JSON 输出转换为易于查看的可视化内容，使用户能够更快地诊断和调试性能不佳的查询。

在 Dev Tools 界面中单击 Search Profiler 选项卡，即可进入 Search Profiler 界面。在界面左侧输入索引名称，指定统一的类型名（这里使用统一的_doc 类型名），并在下方写入一条查询语句，界面右侧即显示出各个分片的执行时间，如图 8.32 所示，从而有利于用户推断搜索引擎的性能负载情况。

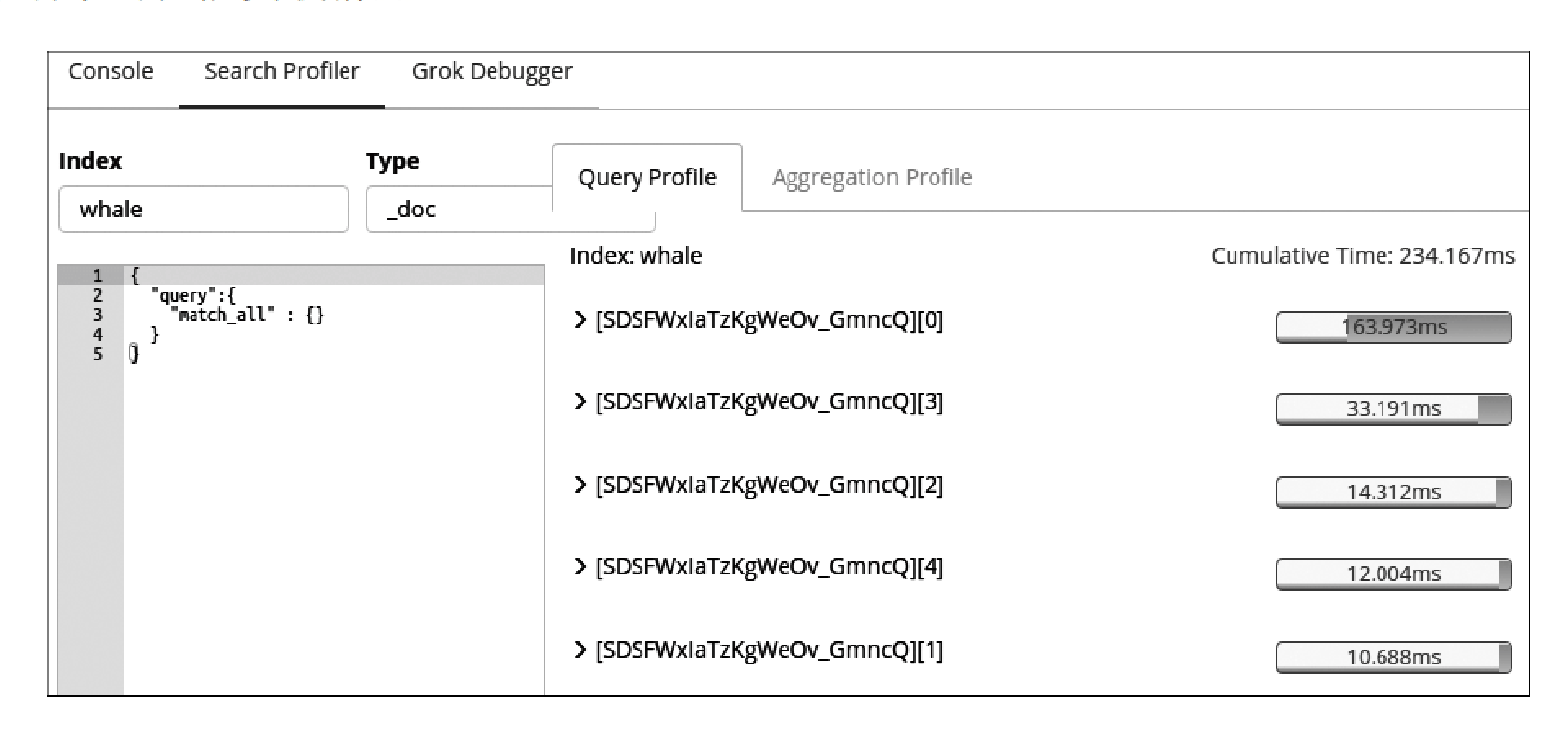


图 8.32 分片执行时间可视化

可执行一个稍复杂的查询，包含一个由全文检索和词项检索组成的布尔查询和一个数值聚合。执行后，界面右侧可以在每个分片中分别查看各种查询的执行时间以及详细的统计数据，如图 8.33 所示。

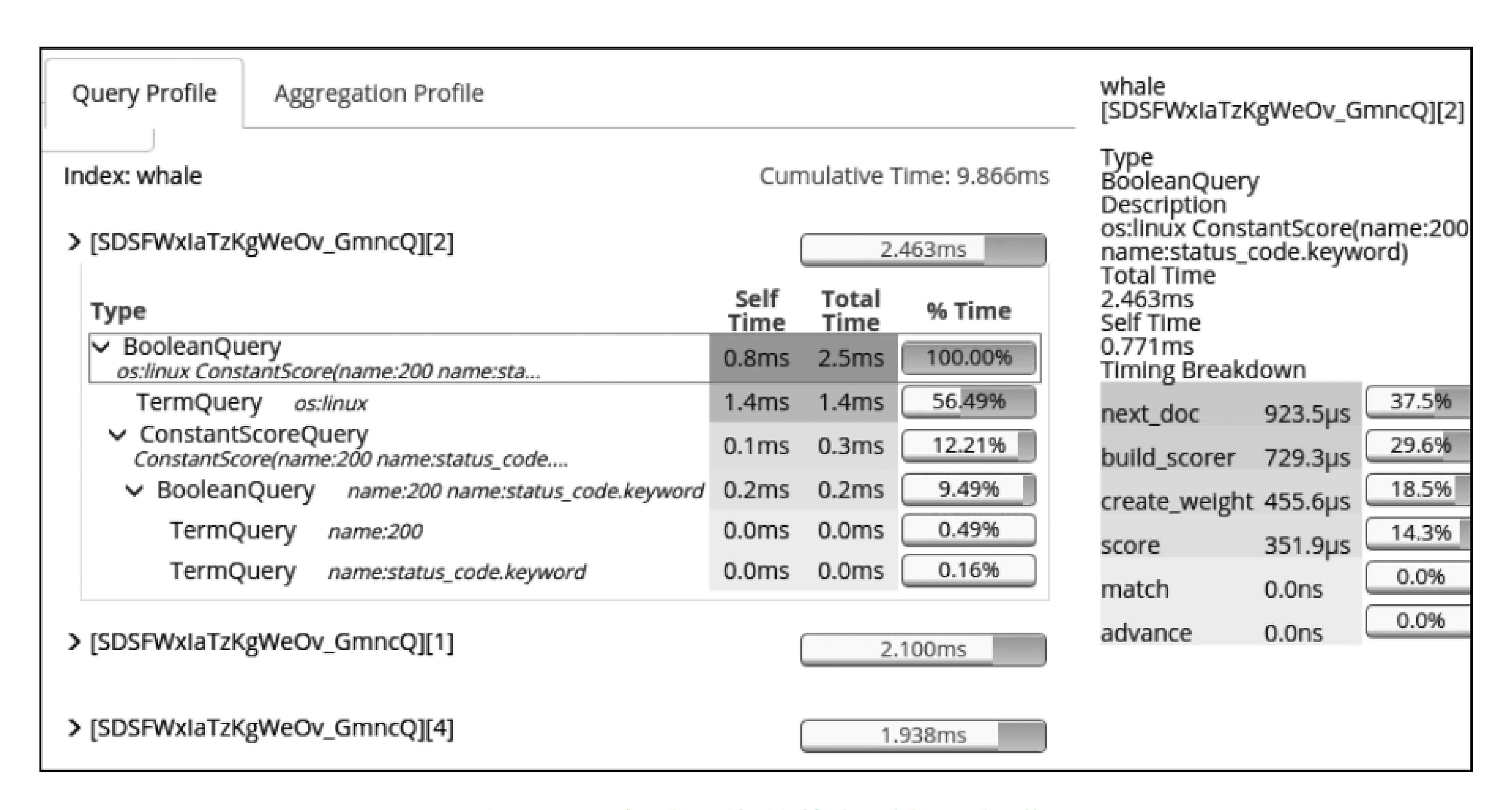


图 8.33 各种查询的执行时间可视化

对于查询中的聚合，单击上方 Aggregation Profile 可以查看聚合对应的相关可视化内容，如图 8.34 所示。

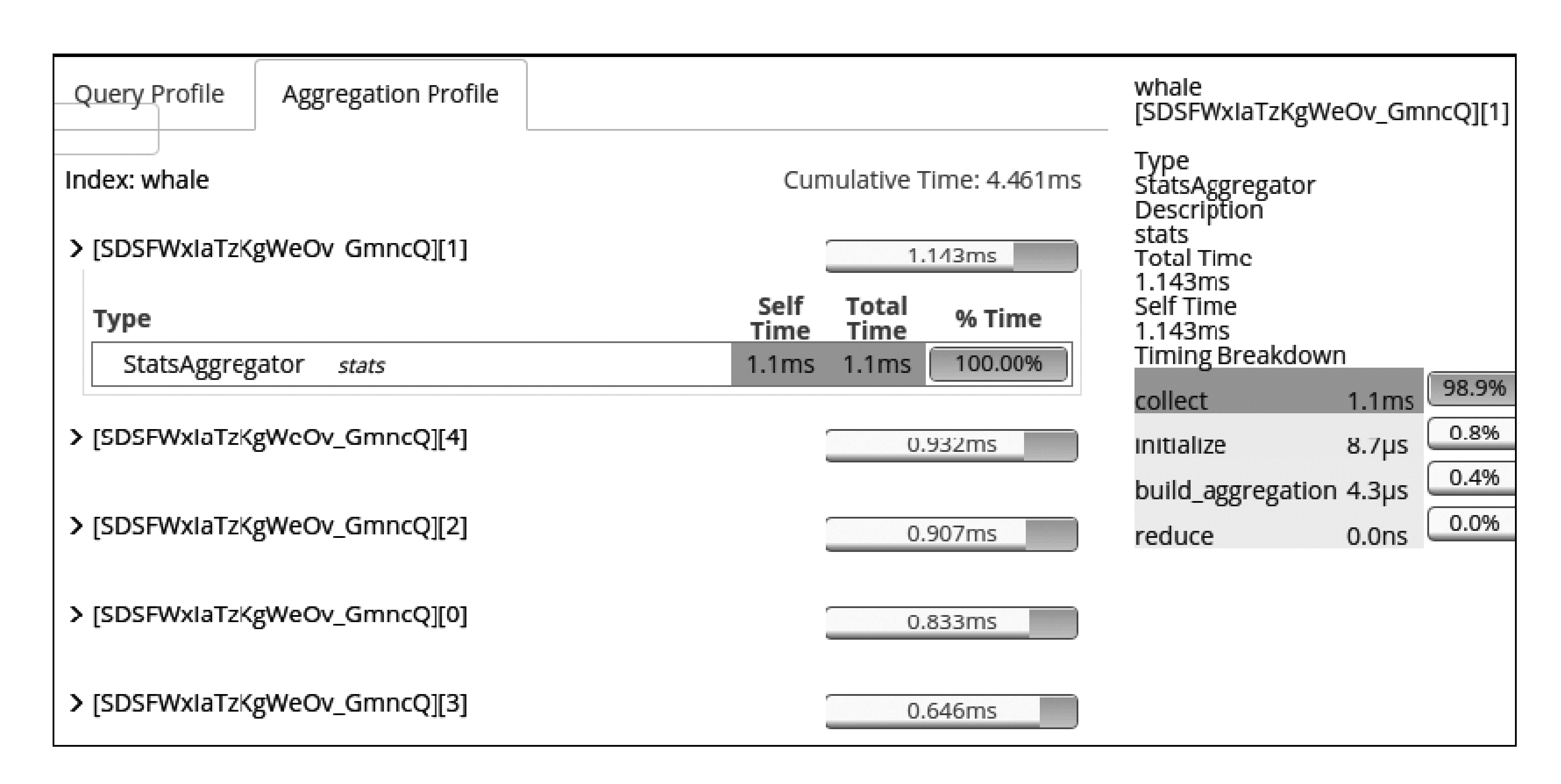


图 8.34　聚合的执行时间可视化

8.10　使用 Grok Debugger 调试 grok 表达式

X-Pack 插件中包含 Grok 调试器工具，用户可以在将 Grok 表达式用于生产环境之前在此工具中构建和调试 Grok 表达式。在 Dev Tools 界面中单击 Grok Debugger 即可跳转到该调试器界面。其使用方法十分简单，只需在 Sample Data 输入框中指定要测试的日志，之后在 Grok Pattern 输入框中写入 Grok 表达式，最后单击 Simulate 按钮，即可对日志信息按照输入的模式进行结构化匹配，如图 8.35 所示。

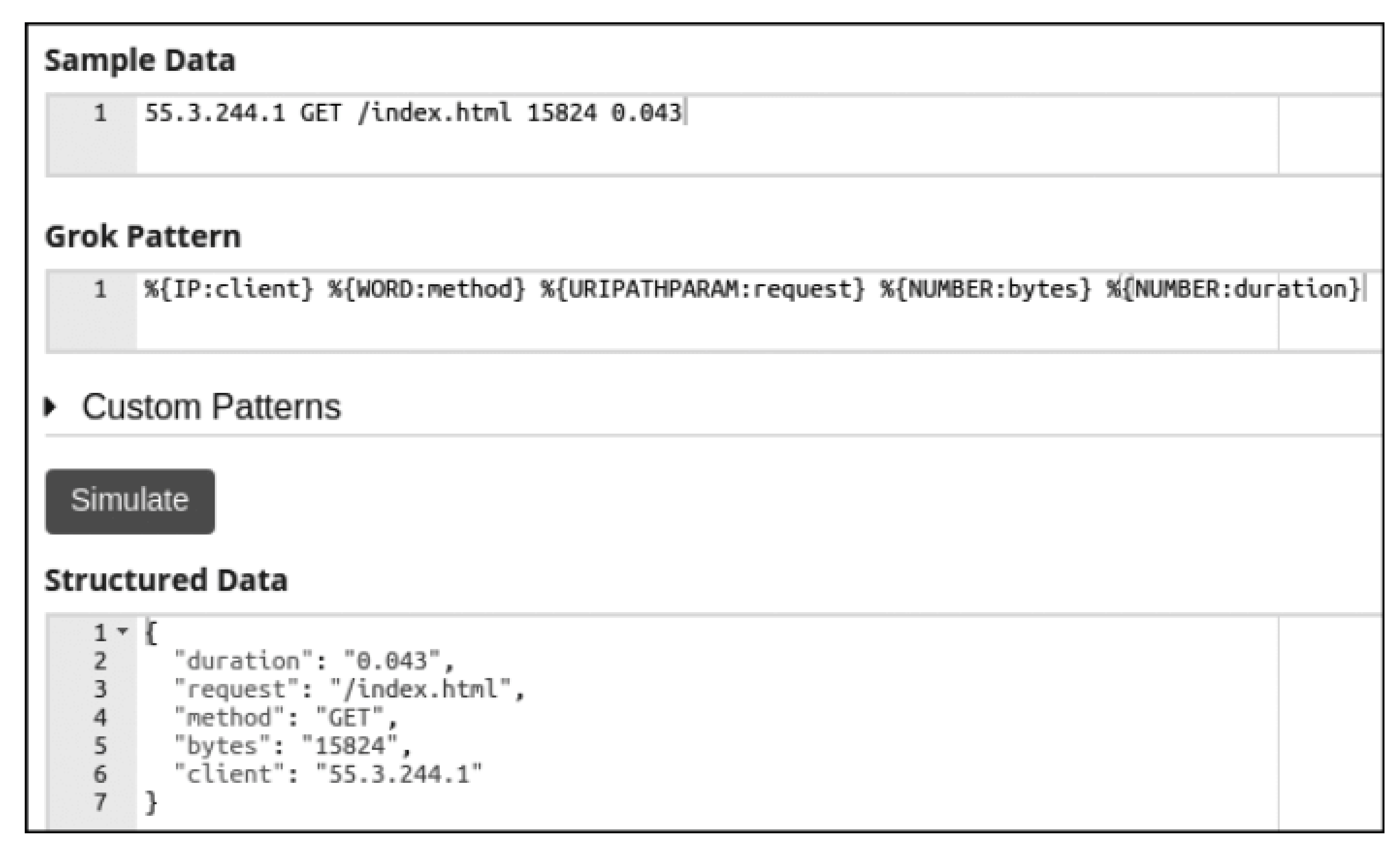


图 8.35　调试 Grok 表达式

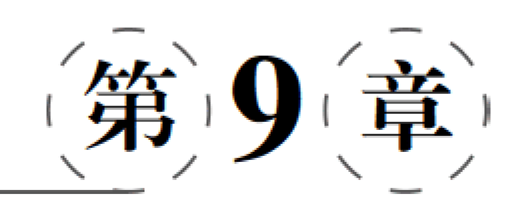
如果用户输入的 Grok 表达式不能实现对日志原文的匹配，或表达式出现任何错误，界面顶端会以弹出红色条幅报错的形式提示给用户。

8.11 扩展知识与阅读

应用软件系统的运行维护工作往往是较为耗时和枯燥的。以往的工作人员需要密切关注众多服务器主机的运行状况，在一些并不优越的工作条件下，需要通过外置的计算机（例如笔记本电脑通过实体接线来连接服务器）了解单台服务器主机的运行数据，并判断其运行状况。在较好的条件下，工作人员可以开发或从其他企业购买成熟的软件系统运行监控管理系统或运维服务。以往的这些工作模式无疑加大了工作人员的工作压力，同时很大程度上抬高了软件服务开发、运行和维护的成本，降低了工作人员的工作效率。针对上述问题，Elastic 公司推出的运行监控插件 X-Pack 能够从应用系统层面上对数据进行监视和问题预判，可以有效地缓解工作人员运维工作烦琐、难度大的问题，同时也节省了传统方式开发或购进运行监控管理系统的开销。

8.12 本章小结

本章对 X-Pack 插件的使用及配置方法进行了描述，内容涉及安全性配置和权限管理、监控集群的运行状态、利用监视器在系统即将出现异常时触发预警操作、为可视化统计图表和检索生成 PDF 报告、通过绘制 Graph 来挖掘数据之间的潜在关联以及利用 Machine Learning 发现数据变化趋势中的异常。X-Pack 的加入可以为 Kibana 的运行提供信息安全和运行维护的保障，同时也为各种可视化结果的序列化操作和对数据中潜在可用信息的挖掘提供了便利。作为以前版本当中各类插件的统一封装包，X-Pack 与 Kibana 的无缝整合加强了系统完整性，从而提升了用户的使用体验。



基于 Beats 的数据解析传输

> Beats is the platform for single-purpose data shippers. They install as lightweight agents and send data from hundreds or thousands of machines to Logstash or Elasticsearch. Beats are great for gathering data. They sit on your servers and centralize data in Elasticsearch. And if you want more processing muscle, Beats can also ship to Logstash for transformation and parsing. The Beats family includes Filebeat, Metricbeat, Packetbeat, Winlogbeat, Auditbeat, Heartbeat, etc.
>
> https://www.elastic.co/products/beats

Beats 是 Elastic 公司推出的一套开源工具，用于将服务器端不同类型的数据进行解析和转换并传输到 Elasticsearch。数据既可以直接传输给 Elasticsearch，也可以通过 Logstash 处理后送入 Elasticsearch。在开源社区中，众多第三方企业、工具或协议均拥有自己的专属 Beats，例如 amazonbeat、apachebeat、httpbeat、mongobeat、mysqlbeat、redisbeat、springbeat、twitterbeat、udpbeat 等。Elastic Stack 官网也针对自己的软件产品开发了 Beats 工具，如 elasticbeat、logstashbeat 等。参照 Elastic Stack 官网资料，Beats 与 Elastic Stack 之间数据传输的示意图如图 9.1 所示。

Elastic 公司推出了 6 种通用 Beats 工具：packetbeat、filebeat、metricbeat、winlogbeat、auditbeat 和 heartbeat。packetbeat 是一种在应用服务器间传输事务信息的分析器，可完成网络数据包传输；filebeat 从服务器端传送日志文件，可完成日志文件数据传输；metricbeat 是一种服务器监控代理程序，可分时段采集服务器上操作系统和服务的各项指标；winlogbeat 负责传输 Windows 事件日志；auditbeat 用来审查系统中用户和进程的活动情况；heartbeat 是一个轻量级守护程序，可安装在远程服务器上，定期检查服务的状态并确定其是否可用。Elastic 公司还推出了 libbeat 运行库，使用 Golang 语言编写。所有的 Beats 工具均使用 libbeat 提供的 API 接口，以执行传输数据到 Elasticsearch、配置数据的输入、实

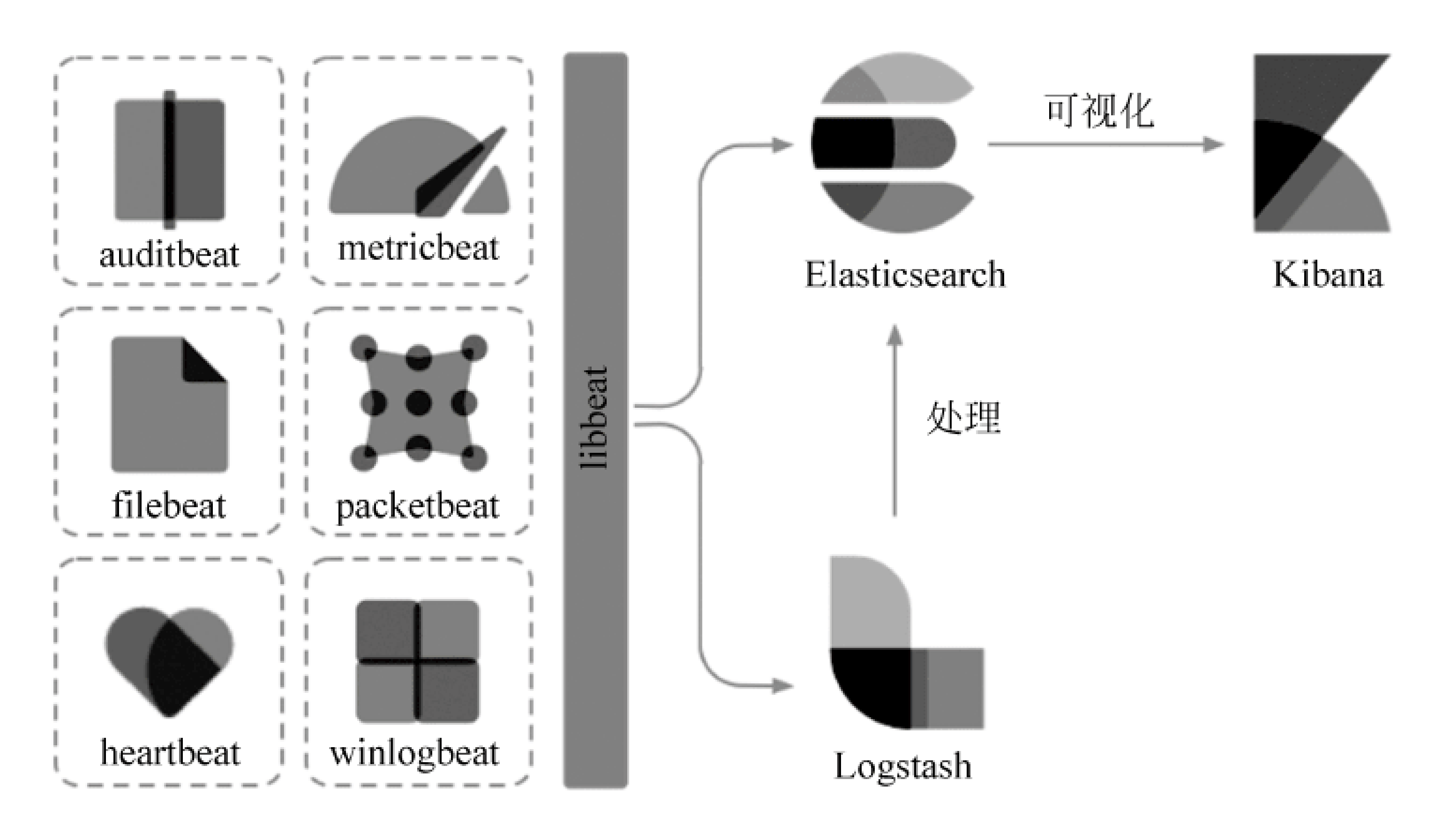


图 9.1　Beats 与 Elastic Stack 间数据传输示意图

现日志记录等任务。限于篇幅，本章只对这 6 种通用 Beats 工具的安装、使用及可视化展示方法进行简要介绍。

9.1 基于 packetbeat 的网络数据包传输

9.1.1 概述

packetbeat 是一种实时网络数据包传输分析工具，可用来与 Elasticsearch 共同构建一套应用程序监控和性能分析系统。packetbeat 的主要功能包括：捕获应用服务器间网络信息流通量，对应用层的数据（如 HTTP、MySQL 或 Redis 等）进行解码、关联请求和响应，以及对每种事务中有价值的输出字段进行记录，等等。packetbeat 能够嗅探服务器间的网络通路，并直接将相关事务信息存入 Elasticsearch 中，这有利于用户对网络信息流通量和日志信息进行分析，也便于用户关注后端程序出现的漏洞或性能缺陷，以完成快速修复。

9.1.2 安装

在安装 packetbeat 之前，需要确保 Elastic Stack 相关产品——Elasticsearch、Logstash（可选）和 Kibana 已经完成安装和配置（Elasticsearch 提供数据的存储和索引功能，Logstash 负责将数据插入 Elasticsearch，Kibana 提供前端可视化展示界面）。

接下来下载和安装 packetbeat。以 64 位 Ubuntu 系统为例，首先确保 libpcap 公用库准备就绪，在终端中执行如下命令来安装 libpcap 0.8（如该公用库已安装，将会输出 libpcap 0.8 已经是最新版的信息）：

```
sudo apt-get install libpcap0.8
```

使用 curl 命令从 Elastic Stack 官网获取 packetbeat 的 DEB 格式安装包(其中 6.2.4 是与当前 Elastic Stack 产品版本一致的版本号,读者可根据实际情况修改):

```
curl -L -O https://artifacts.elastic.co/downloads/beats/packetbeat/packetbeat-6.2.4-amd64.deb
```

获取 packetbeat 安装包的过程如图 9.2 所示。

```
cy@cy-N53SN:~$ curl -L -O https://artifacts.elastic.co/downloads/beats/packetbea
t/packetbeat-6.2.4-amd64.deb
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
 10 13.8M   10 1551k    0     0   265k      0  0:00:53  0:00:05  0:00:48  358k
```

图 9.2　获取 packetbeat 安装包

最后使用下面的终端命令解包并安装 packetbeat:

```
sudo dpkg -i packetbeat-6.2.4-amd64.deb
```

安装过程如图 9.3 所示。

```
cy@cy-N53SN:~$ sudo dpkg -i packetbeat-6.2.4-amd64.deb
[sudo] cy 的密码:
正在选中未选择的软件包 packetbeat。
(正在读取数据库 ... 系统当前共安装有 295815 个文件和目录。)
正准备解包 packetbeat-6.2.4-amd64.deb  ...
正在解包 packetbeat (6.2.4) ...
正在设置 packetbeat (6.2.4) ...
正在处理用于 systemd (229-4ubuntu21.1) 的触发器 ...
正在处理用于 ureadahead (0.100.0-19) 的触发器 ...
ureadahead will be reprofiled on next reboot
cy@cy-N53SN:~$
```

图 9.3　安装 packetbeat

> Tips　这里使用的 DEB 安装包是 Debian Linux 的一种安装格式,能够使用系统自带的包管理器直接自动安装。使用 DEB 包安装应用程序,比其他安装方式更为便捷易用。

9.1.3　配置

使用 DEB 包执行安装后,packetbeat 默认保存在/etc/init.d 目录下,packetbeat 配置文件 packetbeat.yml 默认保存在/etc/packetbeat 目录下,packetbeat 的各项指标可以在该文件中进行配置。下面简要介绍 packetbeat.interfaces.device 配置项及其功能。

packetbeat.interfaces.device 指定安装了 packetbeat 的服务器端与何种设备之间进行信息收发。packetbeat 支持向任何设备均收发信息,该项设置为 any 即可。在配置文件中的协议配置部分可设置每一种协议的端口号,以便 packetbeat 对其进行识别。如需配置某

些标准之外的端口号，也应添加到配置文件中。代码段 9.1 是各类协议和软件产品的端口设置示例。

代码段 9.1：各类协议和软件产品的端口配置

```
packetbeat.protocols:
- type: dns
  ports: [53]
  include_authorities: true
  include_additionals: true
- type: http
  ports: [80, 8080, 8081, 5000, 8002]
- type: memcache
  ports: [11211]
- type: mysql
  ports: [3306]
- type: pgsql
  ports: [5432]
- type: redis
  ports: [6379]
- type: thrift
  ports: [9090]
- type: mongodb
  ports: [27017]
- type: cassandra
  ports: [9042]
- type: tls
  ports: [443]
```

packetbeat 要将数据送入 Elasticsearch，需要指定 Elasticsearch 的 IP 地址和端口信息。代码段 9.2 为 packetbeat.yml 配置文件中指定 Elasticsearch 的 IP 地址和端口信息的实现方法。

代码段 9.2：配置 packetbeat 输出到 Elasticsearch

```
output.elasticsearch:
  hosts: ["localhost:9200"]                    #指定 IP 地址和端口号
```

如果 Elasticsearch 和 Kibana 安装了 X-Pack 并启用了 Security，那么配置中需要带有身份认证信息，如代码段 9.3 所示。

代码段 9.3：带有身份认证信息的 packetbeat 配置

```
output.elasticsearch:
  hosts: ["localhost:9200"]          #指定集群中 Elasticsearch 节点的 IP 地址和端口号
  username: "elastic"                #指定访问 Elasticsearch 集群的用户名
  password: "elastic"                #写入用户的密码
setup.kibana:
  host: "localhost:5601"             #指定 Kibana 实例的 IP 地址和端口号
  username: "elastic"                #指定访问 Kibana 实例的用户名
  password: "elastic"                #写入用户的密码
```

9.1.4　加载索引模板

在 Elasticsearch 中，索引模板用来定义对索引和类型的设置，同时可定义各种文档字段的映射，以决定字段的各项属性。在 packetbeat 中也有类似的索引模板，默认以 YAML 格式保存在/etc/packetbeat 文件夹中，文件名为 fields. yml。packetbeat 成功连接 Elasticsearch 后，将会自动加载这一模板。如果需要让 packetbeat 加载其他索引模板，可在 packetbeat. yml 配置文件中修改 template. name 和 template. path 配置，如代码段 9. 4 所示。

代码段 9.4：修改加载索引模板的配置信息

```
output.elasticsearch:
  hosts: ["localhost:9200"]                  #指定 IP 地址和端口号
  template.name: "packetbeat"                #指定模板名称
  template.fields: "/home/cy/fields.yml"     #指定模板文件路径
  template.overwrite: false                  #指定是否覆盖
```

当这段配置信息中的 template. overwrite 配置为 false 时，即使索引中已存在模板，已有模板也不会被覆盖。如果需要覆盖模板，则应将 template. overwrite：false 修改为 template. overwrite：true。如果要禁用自动加载模板，可不执行代码段 9. 3 所示的配置信息。

> Tips　在 Logstash 的 output 被启用的情况下，自动加载索引模板的功能是不受支持的。

如果需要手动加载模板功能，可执行代码段 9. 5 所示的终端命令。

代码段 9.5：手动加载索引模板功能

```
curl -H 'Content-Type: application/json' -XPUT 'http://localhost:9200/
_template/packetbeat' -d @packetbeat.template.json
```

如果在加载索引模板前已使用 packetbeat 向 Elasticsearch 传输过数据，那么索引文件中可能存在过时数据。在新模板被加载时，可执行代码段 9.6 所示命令删除旧数据，并以此来强制 Kibana 在可视化过程中使用新数据。

代码段 9.6：删除索引文件中的旧数据

```
curl -XDELETE 'http://localhost:9200/packetbeat-*'
```

9.1.5 启动和关闭

在终端中使用如下命令来启动 packetbeat：

```
sudo service packetbeat start
```

终端界面将会输出如图 9.4 所示的日志信息。

```
cy@cy-N53SN:~$ sudo service packetbeat start
[sudo] cy 的密码：
cy@cy-N53SN:~$
```

图 9.4 启动 packetbeat

packetbeat 启动后，即开始获取服务器端网络流通数据。此时可以尝试利用代码段 9.7 所示的 curl 命令来创建一个简单的 HTTP 请求，以便其能被 packetbeat 获取。执行代码段 9.8 所示的命令，可查验该数据是否已被 Elasticsearch 获取。

代码段 9.7：创建网络访问数据

```
curl http://www.elastic.co/> /dev/null
```

代码段 9.8：验证 packetbeat 获取数据的功能

```
curl -H 'Content-Type: application/json' -XGET 'http://localhost:9200/
packetbeat-*/_search?pretty'
```

如需关闭 packetbeat，可以在终端执行如下命令：

```
sudo service packetbeat stop
```

此时终端界面将输出如图 9.5 所示的日志信息。

```
cy@cy-N53SN:~$ sudo service packetbeat stop
cy@cy-N53SN:~$
```

图 9.5　关闭 packetbeat

9.1.6　使用 Kibana 进行可视化展示

packetbeat 在运行时会将收集到的数据存入 Elasticsearch 中以“packetbeat-版本号-日期”开头的索引文件中，此时可在 Kibana 界面中单击 Management 导航按钮，在 Index Patterns 界面中添加这样的索引模式，在索引名称的输入框中填写“packetbeat-*”，待下方出现匹配成功提示后也可以将名称补全，然后在 Time Filter field name 输入框中选择 @timestamp 字段，即可单击 Create index pattern 按钮添加相关索引模式，如图 9.6 所示。

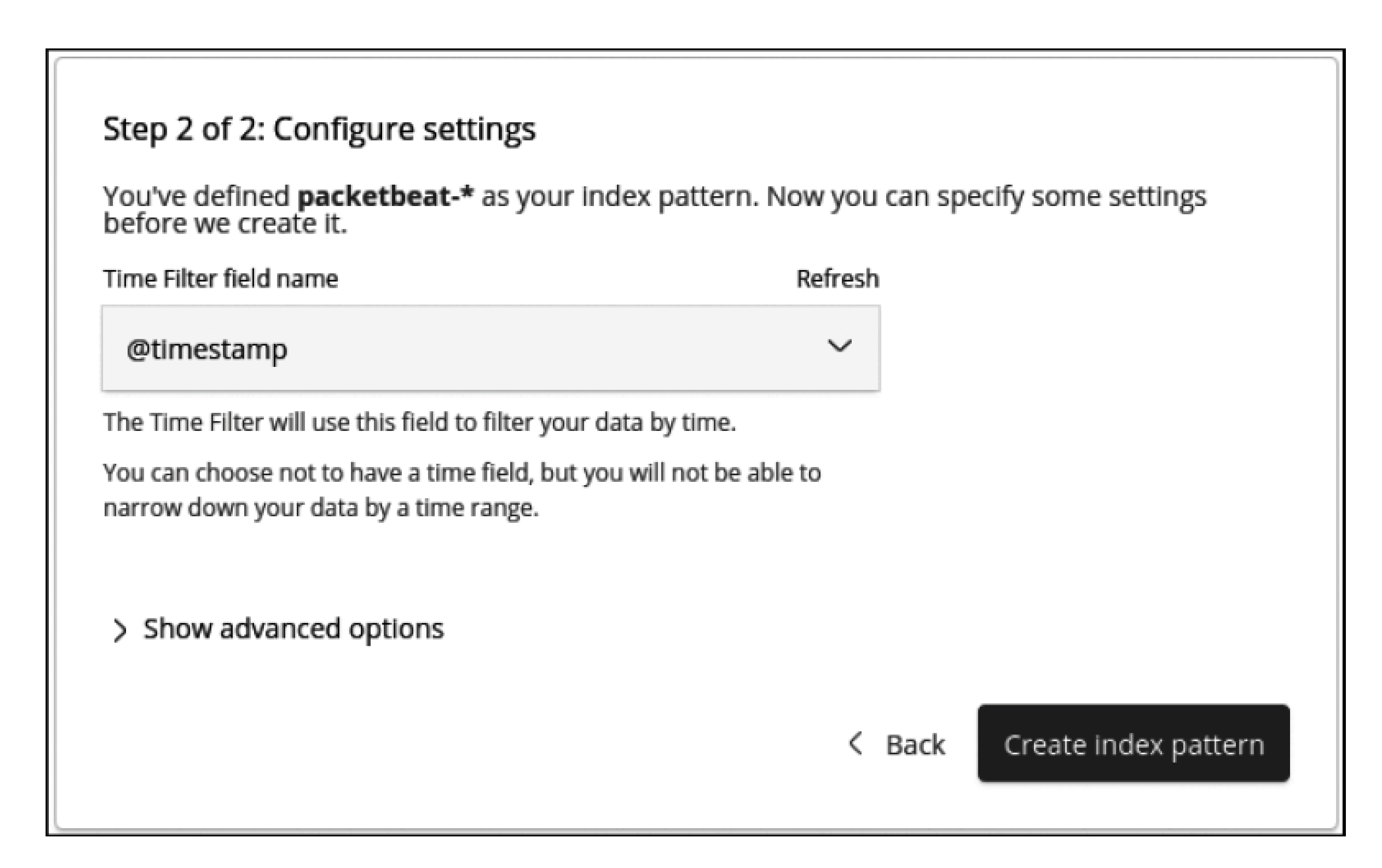


图 9.6　添加索引模式

接下来，可以创建各种可视化统计图表，对索引文件中的数据进行展示。Kibana 中可视化统计图表的创建方法在第 7 章已介绍过，利用其中的方法，可以将 packetbeat 获取的各类数据进行可视化展示。在图 9.7 中，分别对网络流通量、总发包数、传输状态、网络数据包送达总量、数据包传输状态和服务器间 IP 统计等信息进行了展示。其中第一个统计图带有时间轴，显示了网络流通量随时间变化的情况；中间两个统计表显示了数据包和传输状态的统计数据；右侧上方饼图展示了各种网络请求成功和失败的比率，而右侧下方饼图展示了不同 IP 地址之间数据传输的比率。

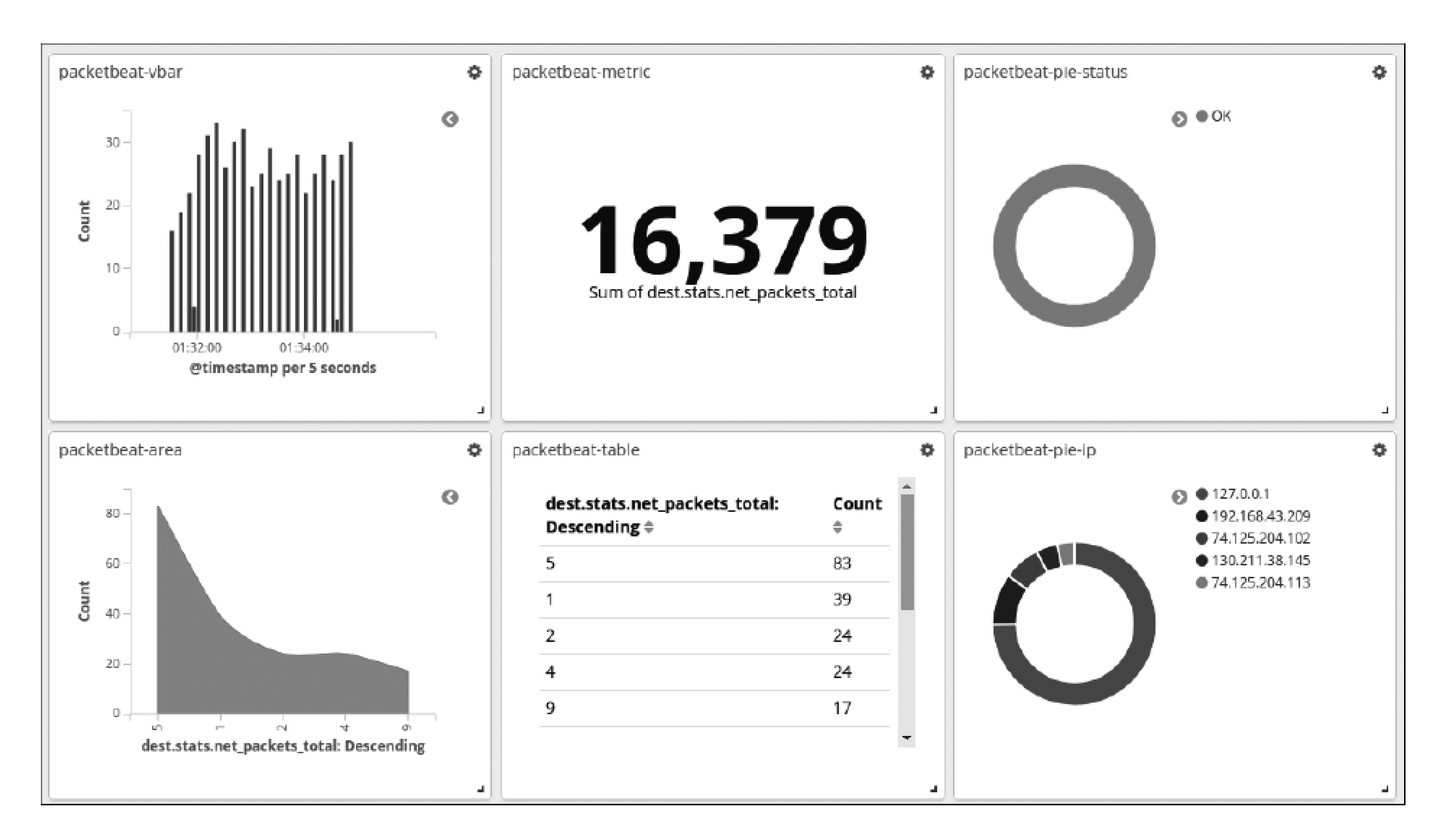


图 9.7 可视化展示

9.2 基于 Filebeat 的日志传输

9.2.1 概述

Filebeat 负责在服务器端传输日志数据，能够对存储日志文件的目录或特定日志文件进行监控，并将数据信息传送到 Elasticsearch 或 Logstash 中。

Filebeat 以一种“探测并传输”的方式工作。程序启动后，将会创建一个或多个探测器（Harvester），以便在特定位置探测日志文件信息并获取日志内容。每一个新发现的日志内容均会被发送给 Spooler 处理程序，该程序将对每一个事件进行聚合，并将聚合之后的数据发送给预先配置好的输出端（如 Elasticsearch、Logstash、Kafka 或 Redis 等）。图 9.8 为来自 ElasticStack 官网的 Filebeat 的数据处理流程示意图。

9.2.2 安装和配置

安装 Filebeat 之前，需要确保 Elastic Stack 相关软件 Elasticsearch、Logstash（可选）和 Kibana 已经完成安装和配置。在 9.1.2 节中已介绍了 libpcap 软件的安装，这里直接使用该软件执行下面的第一个命令，从 Elastic Stack 官网获取 Filebeat 的 DEB 安装包，其中 6.2.4 表示版本号。获取安装包之后，在终端执行第二个命令解压并安装 Filebeat。

```
curl -L -O https://artifacts.elastic.co/downloads/beats/filebeat/filebeat-6.2.4-amd64.deb
```

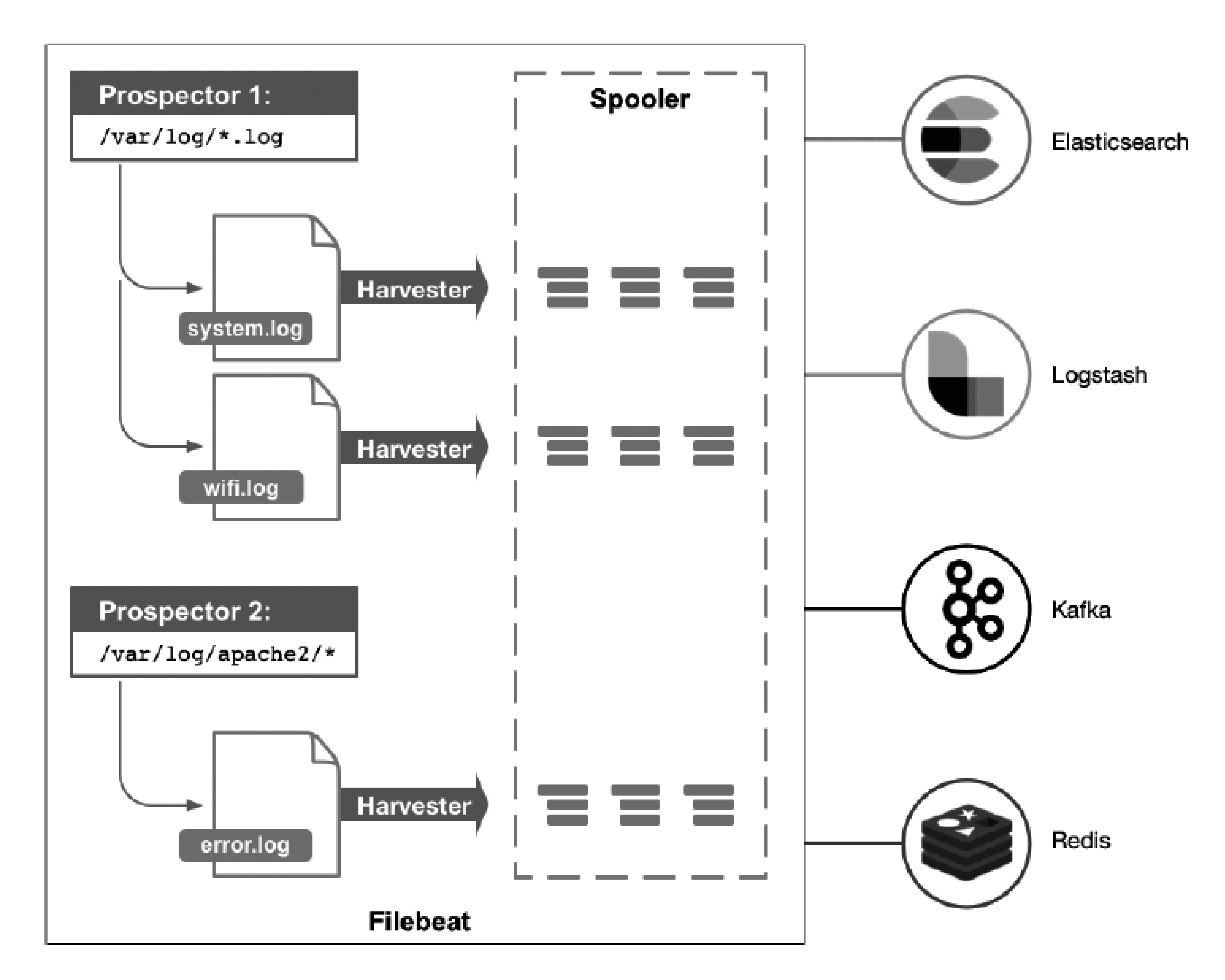
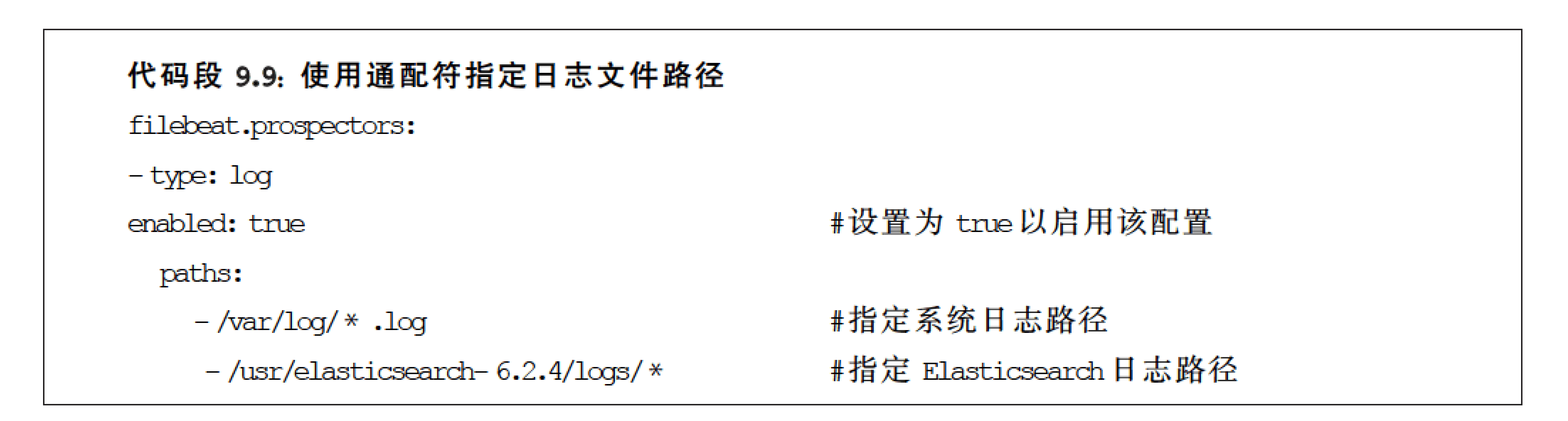


图 9.8　Filebeat 数据处理流程示意图

```
sudo dpkg -i filebeat-6.2.4-amd64.deb
```

Filebeat 程序默认保存在/etc/init.d 目录下，其配置文件 Filebeat.yml 默认保存在/etc/Filebeat 目录下。在 Filebeat.yml 配置文件中，可以对要收集的日志信息所在的路径进行配置。代码段 9.9 演示了在配置文件中使用通配符指定一个或多个日志文件路径的方法。值得注意的是，配置信息中的 enabled 默认为 false(即不应用该 prospectors 配置)，这将导致 Filebeat 组件无法向 Elasticsearch 传输数据。要输出数据到 Elasticsearch 中，则必须将该项的 enabled 设置为 true。


代码段 9.9：使用通配符指定日志文件路径

```
filebeat.prospectors:
-type: log
enabled: true                              #设置为 true 以启用该配置
  paths:
    -/var/log/*.log                        #指定系统日志路径
     -/usr/elasticsearch-6.2.4/logs/*      #指定 Elasticsearch 日志路径
```

如果让 Filebeat 将数据送入 Elasticsearch，可以在 filebeat.yml 中添加代码段 9.10 所示的配置信息，以便指定 Elasticsearch 的 IP 地址和端口号。同时，如果 Elasticsearch 中启用了 X-Pack 的 Security 机制，则需要指定身份认证信息。

代码段 9.10：配置 Filebeat 输出到 Elasticsearch

```
output.elasticsearch:
  hosts: ["localhost:9200"]                #指定 IP 地址和端口号
  username: "elastic"                      #指定用户名
  password: "elastic"                      #指定对应的密码
```

如果需要让 Logstash 额外处理来自 Filebeat 的数据，那么应在 filebeat.yml 配置文件中添加一段指定 Logstash 的 IP 地址和端口号的配置信息，如代码段 9.11 所示，同时不执行代码段 9.10 所示的 Elasticsearch 的配置信息。

代码段 9.11：配置 Filebeat 输出到 Logstash

```
output.logstash:
  hosts: ["localhost:5044"]          #指定 IP 地址和端口号
```


Logstash 在这里默认占用 5044 端口。

Filebeat 的索引模板 filebeat.template.json 默认存放在/etc/filebeat 文件夹中，在输出数据到 Elasticsearch 的功能被启用的情况下，Filebeat 启动时会自动加载默认模板。如果需要加载另一种模板，则需要在 filebeat.yml 配置文件中修改相关配置，如代码段 9.12 所示。

代码段 9.12：修改加载索引模板的配置信息

```
output.elasticsearch:
  hosts: ["localhost:9200"]
  template.name: "filebeat"                    #模板名称
  template.fields: "/home/cy/fields.yml"       #模板文件路径
  template.overwrite: false                    #禁用覆盖
```

如果 Elasticsearch 索引中已经存在一个模板，那么默认不允许覆盖已有的模板，将代码段 9.12 最后一行的 template.overwrite:false 修改为 template.overwrite:true，即可启用覆盖功能。如果要手动加载模板功能，可在终端执行代码段 9.13 所示的 curl 命令。

代码段 9.13：手动加载索引模板功能

```
curl -H 'Content-Type: application/json' -XPUT 'http://localhost:9200/
_template/filebeat' -d @/etc/filebeat/filebeat.template.json
```

代码段 9.14 所示的命令可以用来删除旧数据，并以此来强制 Kibana 在可视化过程中使用新数据。

代码段 9.14：删除索引文件中的旧数据

```
curl -XDELETE 'http://localhost:9200/filebeat-*'
```

9.2.3　启动和关闭

在终端中使用如下命令来启动 Filebeat：

```
sudo service filebeat start
```

终端界面将会输出如图 9.9 所示的日志信息。

```
cy@cy-N53SN:~$ sudo service filebeat start
[sudo] cy 的密码:
cy@cy-N53SN:~$
```

图 9.9　启动 Filebeat

Filebeat 启动后，即开始从预先在配置文件中配置好的位置获取日志信息，并传输到 Elasticsearch 中。如需关闭 Filebeat，可以在终端执行如下命令：

```
sudo service filebeat stop
```

此时终端界面将输出如图 9.10 所示的日志信息。

```
cy@cy-N53SN:~$ sudo service filebeat stop
cy@cy-N53SN:~$
```

图 9.10　关闭 Filebeat

9.2.4　使用 Kibana 进行展示

在 Kibana 的 Management 组件中添加一个新的索引模式，在过滤索引文件名称的输入框中填写 filebeat-* 来匹配所有由 Filebeat 创建的索引文件，然后在下面的 Time-field name 中选择@timestamp，最后单击 Create index pattern 按钮即可添加 Filebeat 的相关索引模式。在 Discover 界面中，可以通过查询来查看目前所有文档的日志数据的统计情况，如图 9.11 所示。

使用条形统计图、时间序列、环形饼图和统计数值等不同的可视化统计图表，对索引文件中不同时段的日志记录数量统计信息进行可视化展示，并将所有可视化结果添加到一个动态仪表板中，如图 9.12 所示。图中的信息表明了在不同的时间段，Filebeat 会占用系统资源传输各种日志信息。其中，左上方的条形图显示了在一定的时间段内日志文件数据被

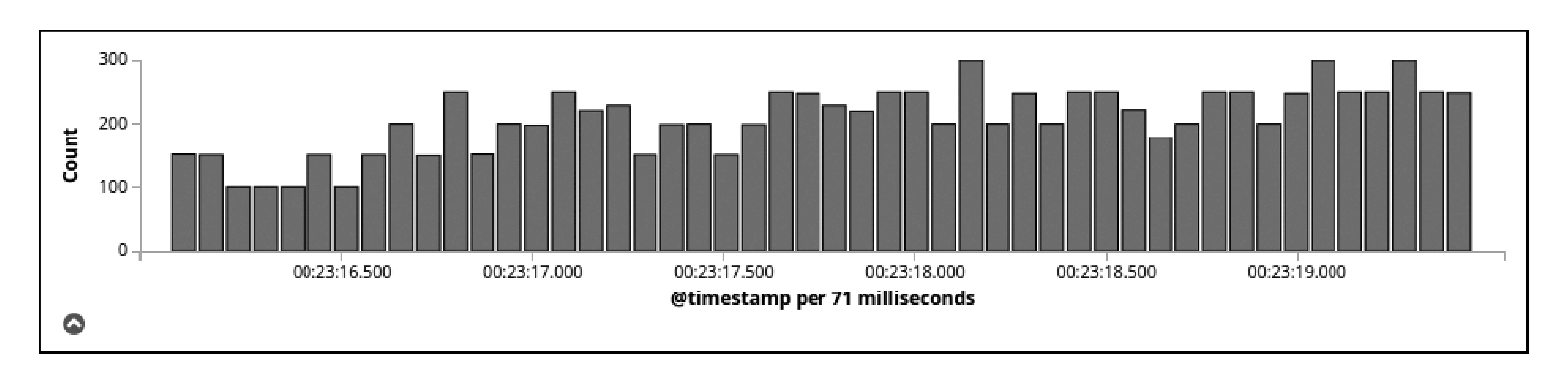


图 9.11　日志数据统计

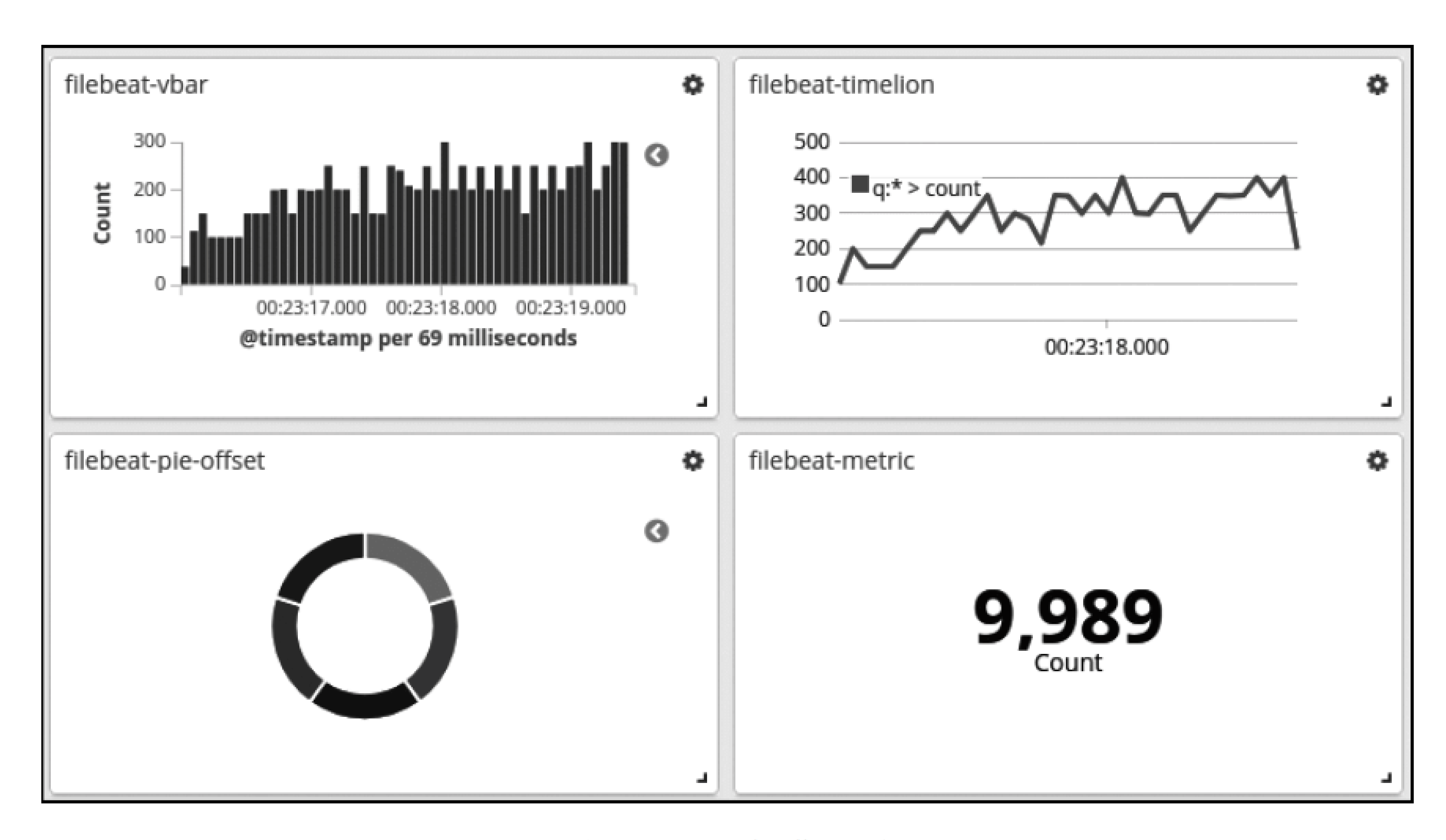


图 9.12　可视化展示

获取和传送的数据量随时间分布情况；右上方的时间序列展示了系统中日志数据获取量趋势折线；左下方的环形饼图表示获取的日志数据所在行号的比率；右下方的统计数值表示 Filebeat 采集到的日志信息数量。

9.3　基于 metricbeat 的系统指标数据传输

9.3.1　概述

metricbeat 是一种轻量级的系统数据指标和统计信息采集器，能够分时段采集服务器上操作系统或正在运行的服务程序中的指标和统计数据。metricbeat 通过收集数据的方式来监控服务器以及 Apache、HAProxy、MongoDB、MySQL、Nginx、PostgreSQL、Redis、System 和 Zookeeper 等服务的运行状况。metricbeat 可以将收集到的数据直接传输到 Elasticsearch 中，也可以将数据发送给 Logstash、Redis 或 Kafka 等。

9.3.2 安装和配置

安装 metricbeat 之前，需要确保 Elastic Stack 相关软件产品 Elasticsearch、Logstash（可选）和 Kibana 已经完成安装和配置。在终端运行下面第一个命令，从 Elastic Stack 官网获取 metricbeat 的 DEB 安装包（其中的 6.2.4 是版本号）。获取安装包之后，在终端执行第二个命令，完成解压并安装 metricbeat。

```
curl -L -O https://artifacts.elastic.co/downloads/beats/metricbeat/metricbeat-6.2.4-amd64.deb
sudo dpkg -i metricbeat-6.2.4-amd64.deb
```

程序默认保存在/etc/init.d 目录下，其配置文件 metricbeat.yml 默认保存在/etc/metricbeat 目录下。在 metricbeat.yml 配置文件中，可以对模块（module）进行定制，以收集特定的指标数据，每一个模块中均定义了一个指标集。代码段 9.15 演示了使用系统运行状态模块来采集服务器系统中 CPU 使用情况、内存使用情况、网络吞吐量等数据指标以及进程的相关统计信息的方法。

代码段 9.15：为 Metricbeat 配置 system 模块

```
metricbeat.modules:
- module: system                    #指定模块的名称
  metricsets:
    - cpu                           #指定模块中具体要采集的指标
    - filesystem
    - memory
    - network
    - process
  enabled: true                     #指定是否启用该模块
  period: 10s                       #指定采集时间间隔
  processes: ['.* ']                #指定进程，这里使用通配符，表示所有进程
  cpu_ticks: false                  #指定是否采集 CPU时钟频率
```

如果让 metricbeat 将数据送入 Elasticsearch，可以在 metricbeat.yml 中添加代码段 9.16 所示的配置信息，需要指定 Elasticsearch 的 IP 地址和端口号；如果 Elasticsearch 已经启用 X-Pack 的 Security 机制，则需要在配置中加入身份验证信息。

代码段 9.16：配置 metricbeat 输出到 Elasticsearch

```
output.elasticsearch:
  hosts: ["localhost:9200"]             #指定 IP地址和端口号
```

```
  username: "elastic"                    #指定用户名
  password: "elastic"                    #指定对应的密码
```

metricbeat 的索引模板 fields.yml 默认存放在/etc/metricbeat 文件夹中。在输出数据到 Elasticsearch 的功能被启用的情况下，metricbeat 启动时会自动加载默认的模板。如果要加载另一种模板，需在 metricbeat.yml 配置文件中修改相关配置，如代码段 9.17 所示。

代码段 9.17：修改加载索引模板的配置信息

```
output.elasticsearch:
  hosts: ["localhost:9200"]
  template.name: "metricbeat"
  template.fields: "/home/cy/fields.yml"
  template.overwrite: false
```

如果 Elasticsearch 索引中已经存在一个模板，默认不允许覆盖已有的模板。将代码段 9.17最后一行的 template.overwrite:false 修改为 template.overwrite:true，即可启用覆盖功能。如果要手动加载模板的功能，可以在终端执行代码段 9.18 所示的 curl 命令。

代码段 9.18：手动加载索引模板功能

```
curl -H 'Content-Type: application/json' -XPUT 'http://localhost:9200/
_template/metricbeat' -d @ /etc/metricbeat/metricbeat.template.json
```

代码段 9.19 所示的命令可以用来删除旧数据，并以此来强制 Kibana 在可视化过程中使用新数据。

代码段 9.19：删除索引文件中的旧数据

```
curl -XDELETE 'http://localhost:9200/metricbeat-*'
```

9.3.3 启动和关闭

在终端中使用如下命令来启动 metricbeat：

```
sudo service metricbeat start
```

终端界面将会输出如图 9.13 所示的日志信息。

```
cy@cy-N53SN:~$ sudo service metricbeat start
[sudo] cy 的密码：
cy@cy-N53SN:~$
```

图 9.13 启动 metricbeat

metricbeat 启动后，即开始采集服务器端操作系统或服务程序的相应数据指标并传输到 Elasticsearch 中。此时可以执行代码段 9.20 所示的 curl 命令，验证服务器端的统计数据是否已被传输到 Elasticsearch 中。

代码段 9.20：测试 metricbeat 的采集和传输功能

```
curl -H 'Content-Type: application/json' -XGET 'http://localhost:9200/
metricbeat-*/_search?pretty'
```

如需关闭 metricbeat，可在终端执行如下命令：

```
sudo service metricbeat stop
```

此时终端界面将输出如图 9.14 所示的日志信息。



图 9.14　关闭 metricbeat

9.3.4　使用 Kibana 进行展示

在 Kibana 的 Management 组件中添加一个新的索引模式，在过滤索引文件名称的输入框中填写 metricbeat-* 来匹配所有由 metricbeat 创建的索引，然后在下面的 Time-field name 中选择@timestamp，单击 Create index pattern 按钮即可添加 metricbeat 的相关索引模式，如图 9.15 所示。

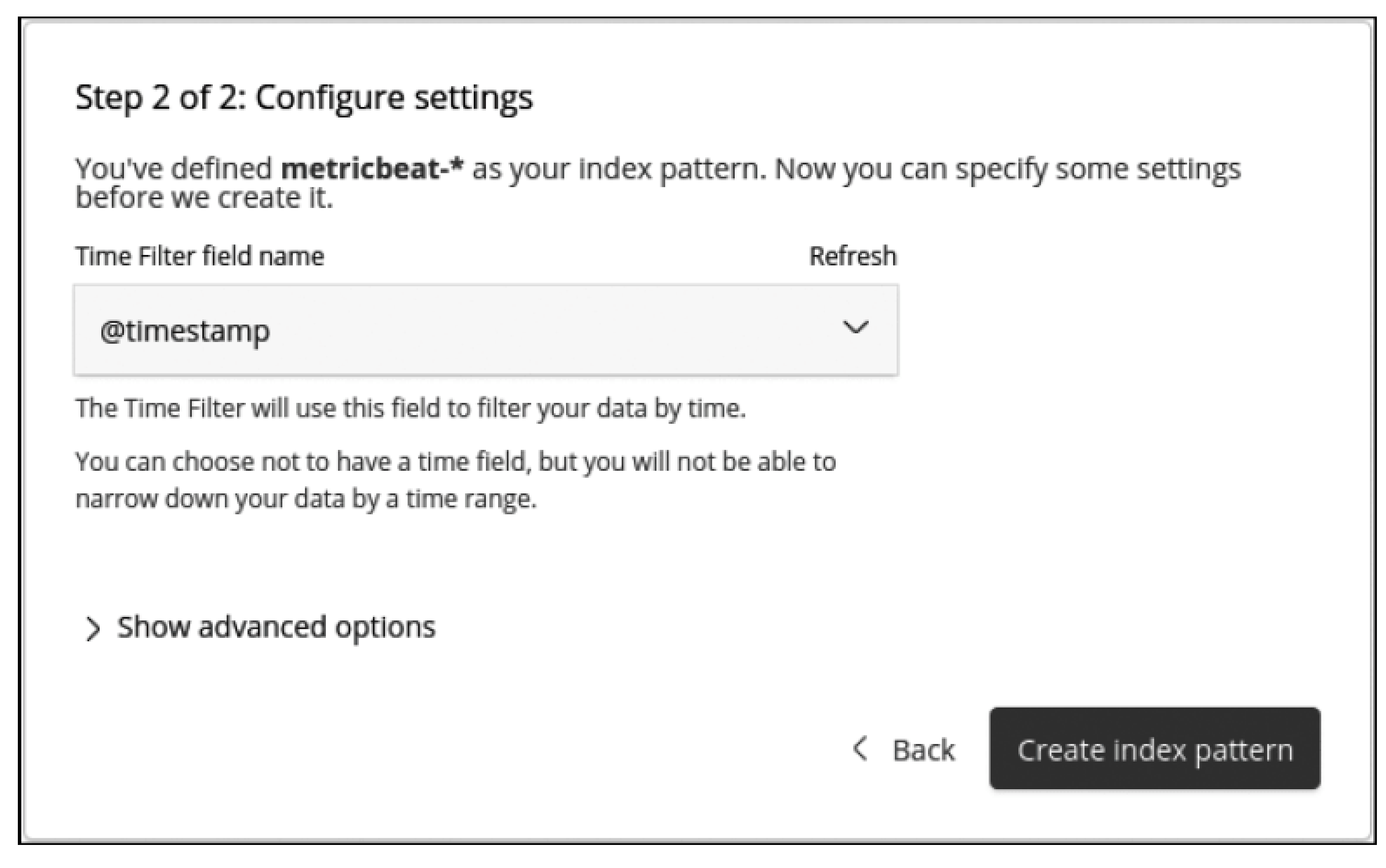


图 9.15　添加索引模式

使用条形统计图、面积图、环形饼图和标签云等不同的可视化统计图表，对索引文件中的数据指标总量、用户名和进程状态、进程名称、挂载设备名称等统计信息进行可视化展示，并将所有可视化结果添加到一个动态仪表板中。图 9.16 中的信息表明了各类系统运行指标数据在总体中的统计量和占比，其中左上方第一个统计数据表示采集到的索引文件中的系统指标总数；中间上方统计表显示了不同状态的用户产生的相关数据的统计数据；右上方标签云反映了这些指标分别来自哪些进程；下方左侧两个统计图反映了不同文件以及不同进程所产生的数据指标统计；右下方饼图则是对不同设备产生的数据指标比率的统计。

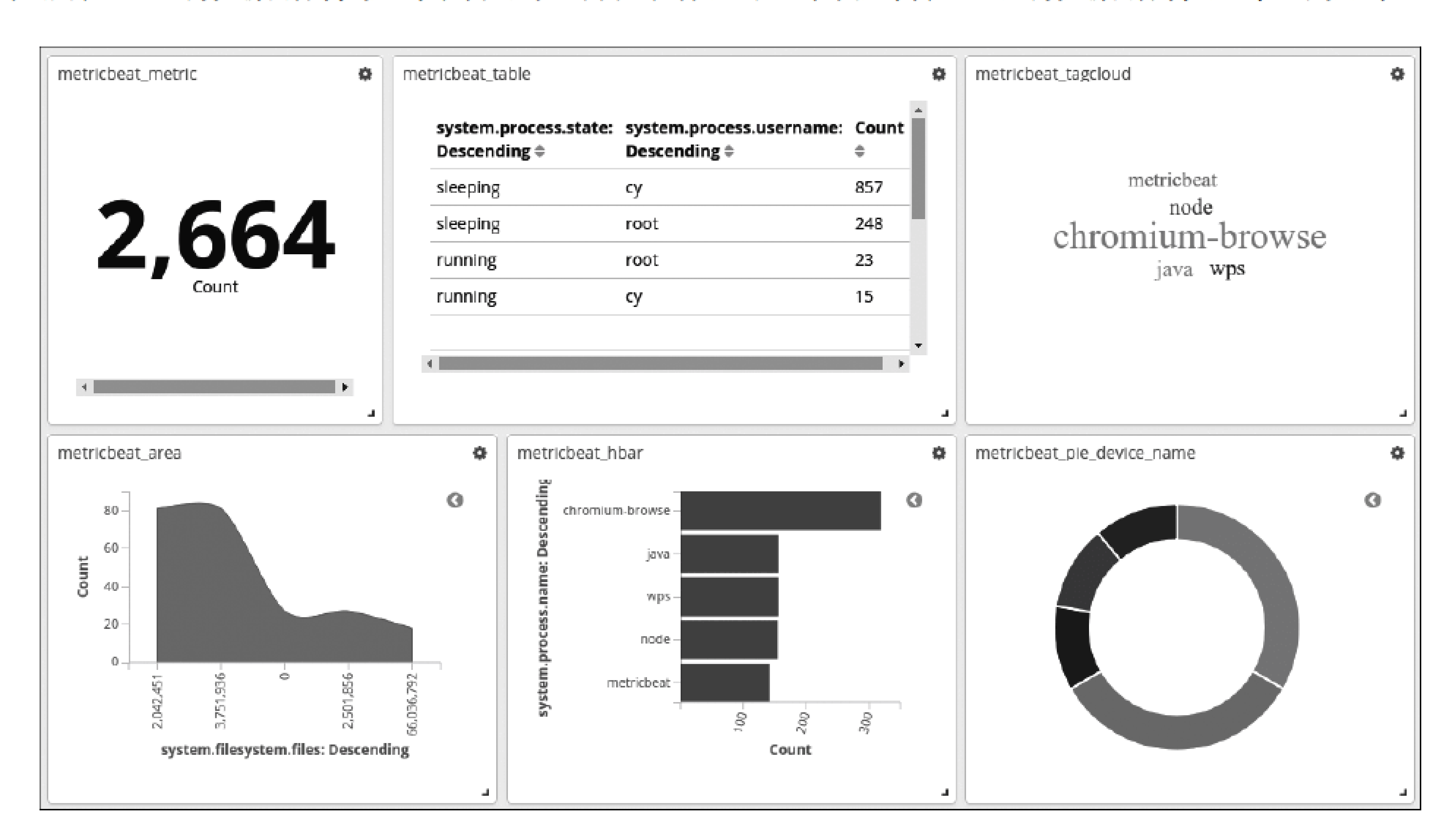


图 9.16　可视化展示

9.4 基于 winlogbeat 的 Windows 事件日志数据传输

9.4.1　概述

winlogbeat 是一种 Windows 服务程序，运行于 Windows XP 及以后版本的操作系统中，负责向 Elasticsearch 或 Logstash 传送 Windows 事件日志数据。winlogbeat 可以对系统中的新事件进行监视，使用 Windows API，从一个或多个事件日志中读取数据，根据用户事先配置好的规则对事件信息进行过滤，然后将事件数据传输至配置好的输出端（如 Elasticsearch 或 Logstash）。每一种事件日志的读取位置信息将被持久化到磁盘中，以便 winlogbeat 重新启动后恢复事件日志位置信息。winlogbeat 可以从正在运行的操作系统中捕获任何类型的事件数据，例如应用程序事件、硬件事件、安全性事件和系统事件等。

9.4.2　安装

winlogbeat 是运行在 Windows 系统中的服务程序。安装 winlogbeat 之前，需确保在 Windows 中已完成 Elastic Stack 相关软件产品 Elasticsearch、Logstash（可选）和 Kibana 的安装和配置。首先，访问 Elastic Stack 官网中的 winlogbeat 下载页面（相应的网页链接为 https://www.elastic.co/downloads/beats/winlogbeat），获取 winlogbeat 程序的 ZIP 包。此时用户需要根据正在使用的 ElasticStack 相关软件的版本（即 Elasticsearch 等程序的版本）和当前 Windows 系统位宽（即 32 位或 64 位）来获取对应版本的 winlogbeat。如果上面的链接打开的页面中提供的版本与正在使用的版本不对应，那么应在页面中单击 past releases 转到历史版本获取页面，在该页面中的产品名称和版本下拉列表中分别选择 winlogbeat 和对应的版本号（这里使用的版本是 6.2.4，如图 9.17 所示）。筛选出对应版本的程序后，单击右侧的 Download 按钮，并选择程序位宽，即可下载程序 ZIP 包。



图 9.17　选择对应版本的 winlogbeat

下载 ZIP 包之后，将其解压到 C:\Program Files 目录中，并将解压之后的目录重命名为 winlogbeat。以管理员身份运行 Windows PowerShell 窗口，然后即可在其中执行安装命令。PowerShell 拥有“执行策略”的配置，默认为 Restricted，表示不允许任何脚本执行，包括安装 winlogbeat 需要执行的 ps1 文件（微软，2017）。安装前需要修改 PowerShell 的执行策略以允许执行脚本命令。此时需要在 PowerShell 窗口中执行以下 Set-ExecutionPolicy 命令将执行策略改为 UnRestricted，即允许未签名的脚本运行。

```
Set-ExecutionPolicy -ExecutionPolicy UnRestricted
```

修改 PowerShell 的执行策略之后，即可安装 winlogbeat。执行下面的命令，进入存放 winlogbeat 安装文件的目录，并执行配置文件 install-service-winlogbeat.ps1。

```
cd 'C:\Program Files\Winlogbeat'
.\install-service-winlogbeat.ps1
```

出于系统安全性考虑，在完成安装之后，应执行如下命令将 PowerShell 的执行策略改

回默认的 Restricted：

```
Set-ExecutionPolicy -ExecutionPolicy Restricted
```

winlogbeat 的安装过程如图 9.18 所示。

```
管理员: Windows PowerShell
Windows PowerShell
版权所有 (C) 2016 Microsoft Corporation。保留所有权利。

PS C:\windows\system32> Set-ExecutionPolicy -ExecutionPolicy Unrestricted

执行策略更改
执行策略可帮助你防止执行不信任的脚本。更改执行策略可能会产生安全风险，如 http://go.microsoft.com/fwlink/?LinkID=135170
中的 about_Execution_Policies 帮助主题所述。是否要更改执行策略?
[Y] 是(Y)  [A] 全是(A)  [N] 否(N)  [L] 全否(L)  [S] 暂停(S)  [?] 帮助 (默认值为“N”): Y
PS C:\windows\system32> cd 'C:\Program Files\winlogbeat-6.2.4-windows-x86_64'
PS C:\Program Files\winlogbeat-6.2.4-windows-x86_64> .\install-service-winlogbeat.ps1

Status   Name           DisplayName
------   ----           -----------
Stopped  winlogbeat     winlogbeat


PS C:\Program Files\winlogbeat-6.2.4-windows-x86_64> Set-ExecutionPolicy -ExecutionPolicy Restricted

执行策略更改
执行策略可帮助你防止执行不信任的脚本。更改执行策略可能会产生安全风险，如 http://go.microsoft.com/fwlink/?LinkID=135170
中的 about_Execution_Policies 帮助主题所述。是否要更改执行策略?
[Y] 是(Y)  [A] 全是(A)  [N] 否(N)  [L] 全否(L)  [S] 暂停(S)  [?] 帮助 (默认值为“N”): Y
PS C:\Program Files\winlogbeat-6.2.4-windows-x86_64> _
```

图 9.18　安装 winlogbeat 服务

> Tips　如果当前使用的操作系统是 Windows XP，那么用户需要事先下载并安装 Windows PowerShell 程序。

9.4.3　配置

winlogbeat 的配置文件 winlogbeat.yml 默认保存在解压后的 C:\Program Files\Winlogbeat 文件夹中，可以对采集事件日志的类型、输出端主机地址和输出日志等进行配置。对于事件日志类型，配置文件中默认设置了 Application、Security 和 System 3 种类型，如代码段 9.21 所示。

代码段 9.21：事件日志配置

```
winlogbeat.event_logs:
  - name: Application
  - name: Security
  - name: System
```

除以上 3 种事件日志类型之外，还可以在配置列表中添加其他类型，Windows 中的事件日志类型可以通过执行如下 PowerShell 命令来查看：

```
Get-EventLog *
```

命令的执行结果如图 9.19 所示，图中最右列 Log 中的信息即为可用的事件日志类型。

```
管理员: Windows PowerShell
Windows PowerShell
版权所有 (C) 2016 Microsoft Corporation。保留所有权利。

PS C:\windows\system32> Get-Eventlog *

  Max(K) Retain OverflowAction        Entries Log
  ------ ------ --------------        ------- ---
  20,480      0 OverwriteAsNeeded      21,323 Application
  20,480      0 OverwriteAsNeeded           0 HardwareEvents
     512      7 OverwriteOlder              0 Internet Explorer
  20,480      0 OverwriteAsNeeded           0 Key Management Service
     128      0 OverwriteAsNeeded         109 OAlerts
  20,480      0 OverwriteAsNeeded      25,876 Security
  20,480      0 OverwriteAsNeeded      22,183 System
  15,360      0 OverwriteAsNeeded          44 Windows PowerShell

PS C:\windows\system32>
```

图 9.19　获取事件日志列表

对于 winlogbeat 向外传输的配置，代码段 9.22 演示了 winlogbeat 向 Elasticsearch 传输事件日志信息的配置，其中以数组的形式提供了 Elasticsearch 所在主机 IP 地址和端口号两项网络位置信息。

代码段 9.22：传输数据到 Elasticsearch 的配置

```
output.elasticsearch:
  hosts: ["localhost:9200"]
```

如果需要设置 winlogbeat 向 Logstash 传输数据以进行对数据的额外处理，可以在配置文件中添加代码段 9.23 所示的配置信息，在数组中指定 Logstash 的 IP 地址和端口号，同时将代码段 9.22 所示的向 Elasticsearch 输出数据的配置信息注释掉即可。

代码段 9.23：传输数据到 Logstash 的配置

```
output.logstash:
  hosts: ["127.0.0.1:5044"]
```

Tips　自动加载模板的功能仅对 Elasticsearch 可用，在配置 winlogbeat 输出事件日志数据到 Logstash 时，需要手动更改 winlogbeat 的索引模板配置。

对于输出日志文件的配置，可以在配置文件中设定是否将 winlogbeat 生成的日志信息输出为日志文件、输出日志文件的保存路径、日志信息记录的级别等配置信息。代码段 9.24 演示了将 INFO 级别的日志信息以日志文件的形式输出到 C:\rogramData\winlogbeat\

Logs 目录的方法。

代码段 9.24：输出日志配置

```
logging.to_files: true                              #指定是否输出日志到文件
logging.files:
  path: C:\ProgramData\winlogbeat\Logs              #指定输出日志的位置
logging.level: info                                 #指定日志信息为 INFO 级别
```

Tips 日志一般分为 5 个级别：DEBUG、INFO、WARN、ERROR 和 FATAL。DEBUG 用于调试过程中了解变量的值等信息，INFO 用于向软件用户输出一般提示信息，WARN 用于输出警告提示信息，ERROR 用于输出错误提示信息，FATAL 用于输出严重错误提示信息。

要对当前配置文件中的配置信息进行测试，可以在 PowerShell 窗口中执行如下命令：

```
.\winlogbeat.exe test config -c .\winlogbeat.yml -e
```

窗口中将会输出关于当前配置的各项基本信息，如图 9.20 所示。

```
管理员: Windows PowerShell
Windows PowerShell
版权所有 (C) 2016 Microsoft Corporation。保留所有权利。

PS C:\windows\system32> cd 'C:\Program Files\winlogbeat-6.2.4-windows-x86_64\'
PS C:\Program Files\winlogbeat-6.2.4-windows-x86_64> .\winlogbeat.exe test config -c .\winlogbeat.yml -e
2018-07-16T17:23:06.027+0800    INFO    instance/beat.go:468    Home path: [C:\Program Files\winlogbeat-6.2.4-windows-x86_64] Config path: [C:\Program Files\winlogbeat-6.2.4-windows-x86_64] Data path: [C:\Program Files\winlogbeat-6.2.4-windows-x86_64\data] Logs path: [C:\Program Files\winlogbeat-6.2.4-windows-x86_64\logs]
2018-07-16T17:23:06.040+0800    INFO    instance/beat.go:475    Beat UUID: 7f505739-113d-4500-aa78-f292fb8a30da
2018-07-16T17:23:06.040+0800    INFO    instance/beat.go:213    Setup Beat: winlogbeat; Version: 6.2.4
2018-07-16T17:23:06.041+0800    INFO    elasticsearch/client.go:145     Elasticsearch url: http://localhost:9200
2018-07-16T17:23:06.041+0800    INFO    pipeline/module.go:76   Beat name: cy-PC
2018-07-16T17:23:06.042+0800    INFO    beater/winlogbeat.go:56 State will be read from and persisted to C:\Program Files\winlogbeat-6.2.4-windows-x86_64\data\.winlogbeat.yml
Config OK
PS C:\Program Files\winlogbeat-6.2.4-windows-x86_64> _
```

图 9.20　输出当前配置文件中的基本信息

winlogbeat 的索引模板 winlogbeat.template.json 默认存放在解压后的 C:\Program Files\Winlogbeat 目录中，在输出数据到 Elasticsearch 的功能被启用的情况下，winlogbeat 启动时会自动加载默认的模板。如果需要加载另一种模板，则需要在 winlogbeat.yml 配置文件中修改相关配置，如代码段 9.25 所示。

代码段 9.25：修改加载索引模板的配置信息

```
output.elasticsearch:
  hosts: ["localhost:9200"]
```

```
template.name: "winlogbeat"
template.fields: "C:\Users\cy\Documents\fields.yml"
template.overwrite: false
```

如果 Elasticsearch 索引文件中已经存在一个模板，那么默认不允许覆盖已有的模板，将代码段 9.25 最后一行的 template.overwrite: false 修改为 template.overwrite: true 即可启用覆盖功能。要手动加载模板的功能，可以在 PowerShell 中执行代码段 9.26 所示的命令。

代码段 9.26：手动加载索引模板功能

```
Invoke - RestMethod - Method Put - ContentType "application/json" - InFile
winlogbeat.template.json - Uri
http://localhost:9200/_template/winlogbeat- 6.2.4
```

代码段 9.27 所示的命令可以用来删除旧数据，并以此来强制 Kibana 在可视化过程中使用新数据。

代码段 9.27：删除索引文件中的旧数据

```
Invoke - RestMethod - Method Delete "http://localhost:9200/winlogbeat- * "
```

9.4.4　启动和关闭

进入 winlogbeat 所在目录，在 PowerShell 窗口中使用如下命令来启动 winlogbeat：

```
Start- Service winlogbeat
```

启动后，winlogbeat 即开始从预先在配置文件中配置好的位置获取日志信息，并传输到 Elasticsearch 中。

winlogbeat 启动后，可以在 PowerShell 窗口执行 services.msc 命令打开 Windows 服务窗口，并在其中查看 winlogbeat 的运行状态。

从图 9.21 中不难看出，winlogbeat 服务已经出现在服务程序列表中，其详细属性如图 9.22 所示，其中 winlogbeat 的服务状态显示“正在运行”。

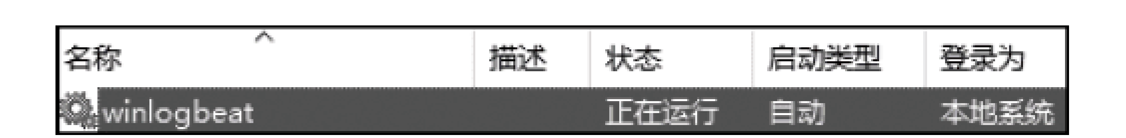


图 9.21　winlogbeat 出现在服务程序列表中

如需关闭 winlogbeat，可以在 PowerShell 窗口执行 Stop-Service winlogbeat 命令。

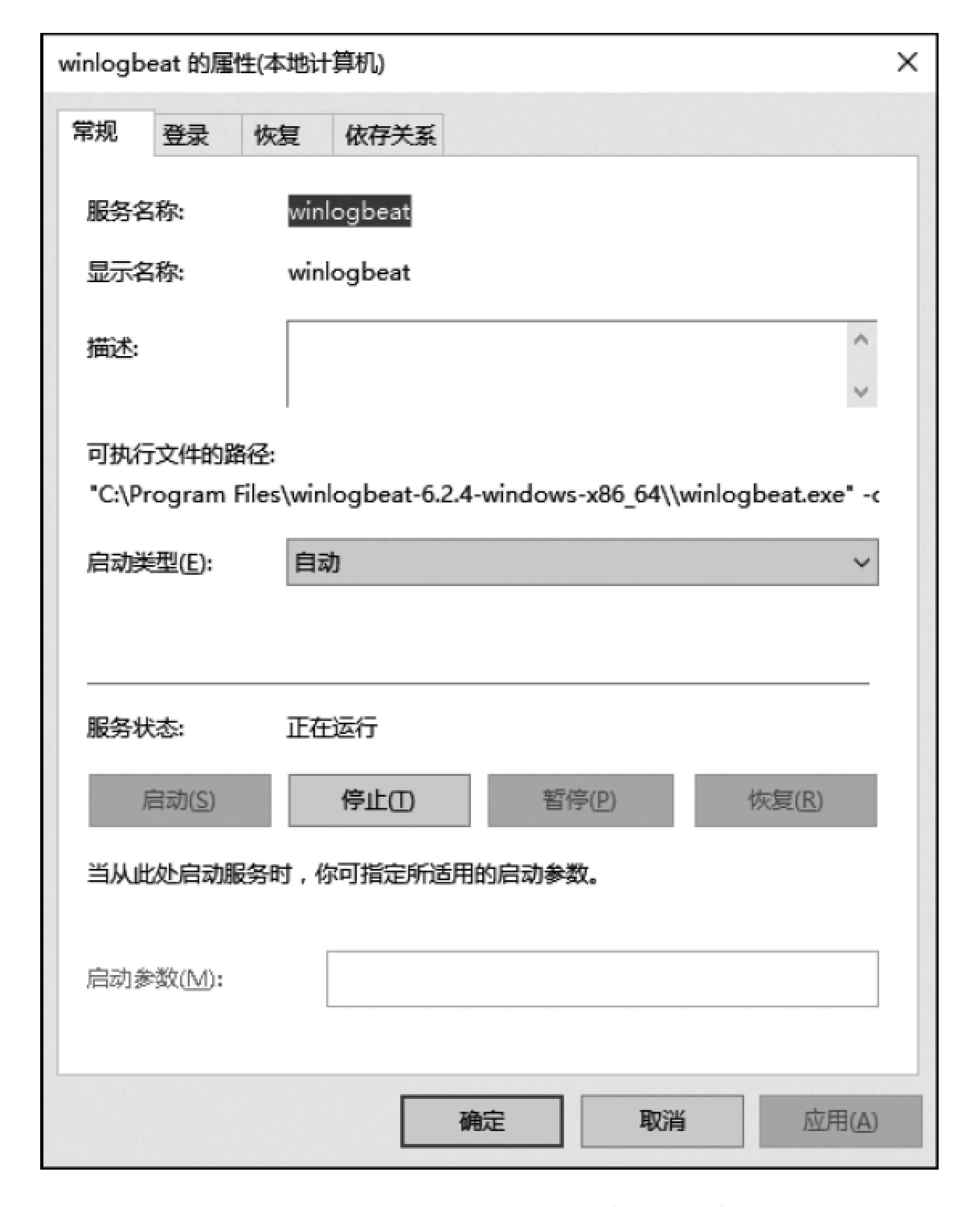


图 9.22　winlogbeat 服务的属性

9.4.5　使用 Kibana 进行展示

在 Kibana 的 Management 组件中添加一个新的索引模式，在过滤索引文件名称的输入框中填写 winlogbeat-* 来匹配所有由 winlogbeat 创建的索引文件，然后在下面的 Time-field name 中选择@timestamp，单击 Create index pattern 按钮，即可添加 winlogbeat 相关索引模式，如图 9.23 所示。

使用条形统计图、表格、饼图、统计数值和数据表等不同的可视化统计图表，可对索引文档中的事件日志总量、日志中反映的服务启动情况、事件来源、事件类型等统计信息进行可视化展示，并将所有可视化结果添加到一个动态仪表板中。图 9.24 展示了 Windows 系统中各类事件日志数据的统计情况，上方左侧的条形图反映了不同时间段系统事件日志记录数量统计；上方中间的数据表统计了系统事件日志中各级别日志信息的数量；上方右侧的统计数值表示事件日志和事件编号的总数；下方的 3 个饼图分别对系统安全性事件、事件日志的类型以及事件来源各部分的比率进行了统计。

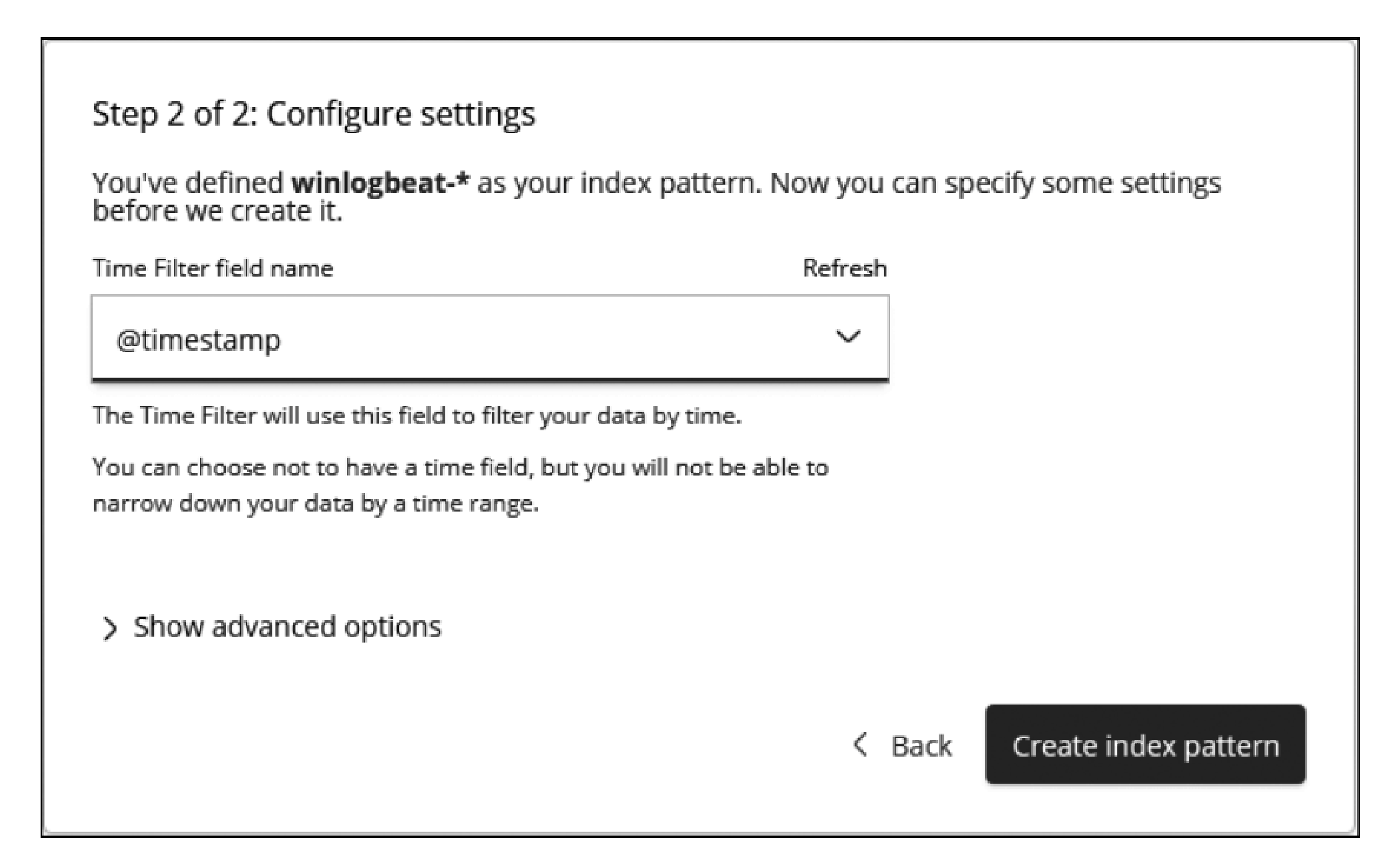


图 9.23　添加索引模式

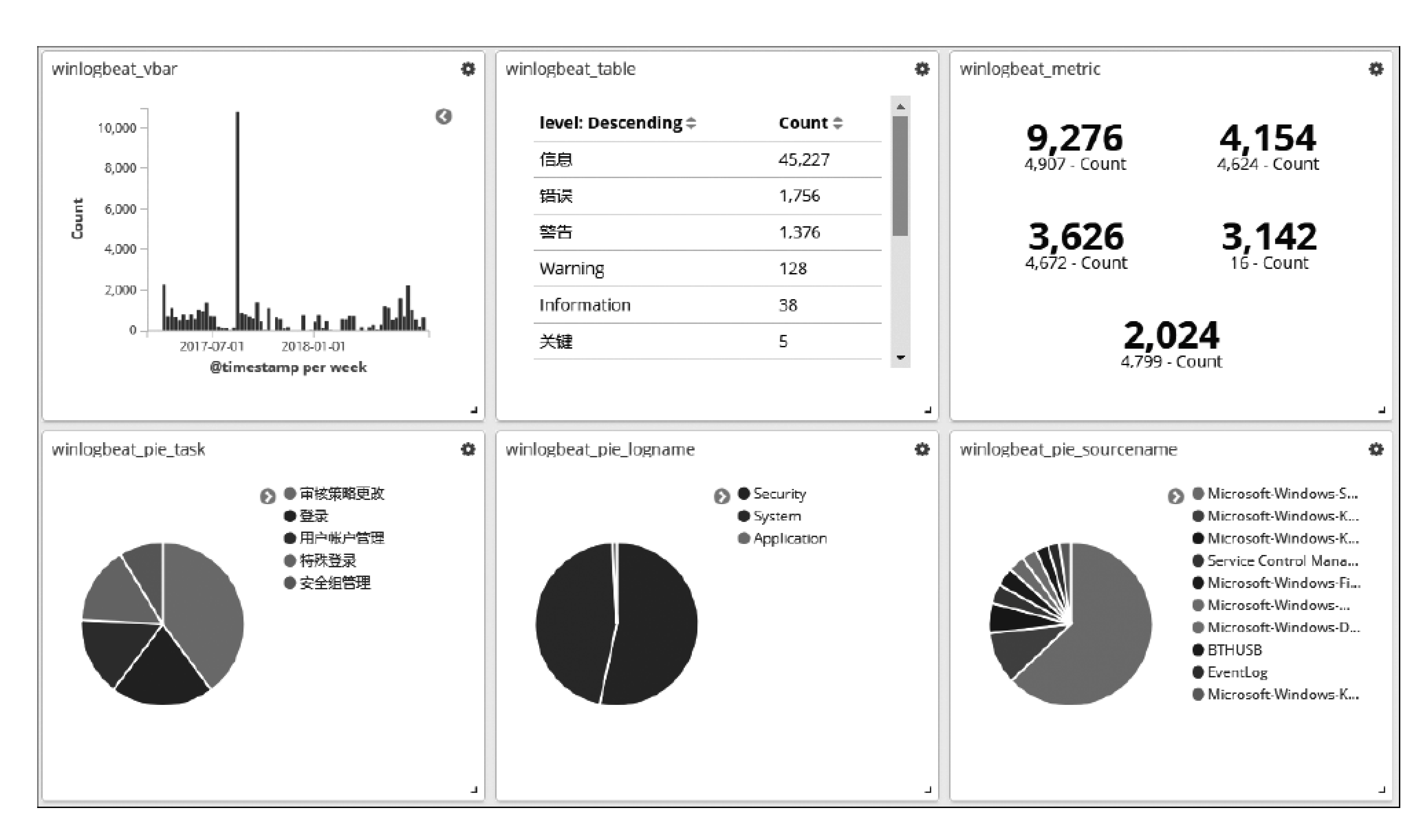


图 9.24　可视化展示

9.5 基于 auditbeat 的用户和进程活动审计

9.5.1 概述

auditbeat 是一个轻量级的数据传输工具，可以用来审计系统上用户和进程的活动。例如，auditbeat 可以从 Linux Audit Framework 收集和汇聚审计事件，还可以检测关键文件（如二进制文件和配置文件）的更改，并识别潜在的安全策略违规事件。

9.5.2 安装和配置

安装 auditbeat 之前，需要确保 Elastic Stack 相关软件产品 Elasticsearch、Logstash（可选）和 Kibana 已经完成安装和配置。在终端运行下面第一个命令，从 Elastic Stack 官网获取 auditbeat 的 DEB 安装包（其中的 6.2.4 是版本号）。获取安装包之后，在终端执行第二个命令，完成解压并安装 auditbeat。

```
curl -L -O https://artifacts.elastic.co/downloads/beats/auditbeat/auditbeat-
6.2.4-amd64.deb
sudo dpkg -i auditbeat-6.2.4-amd64.deb
```

程序默认保存在/etc/init.d 目录下，其配置文件 auditbeat.yml 默认保存在/etc/auditbeat 目录下。在 auditbeat.yml 配置文件中，可以对模块（module）进行定制，以收集特定的指标数据，在每一个模块中均定义了一个指标集。代码段 9.28 演示了使用文件完整性模块采集来自服务器系统中多个目录的相关统计信息的方法。

代码段 9.28：为 auditbeat 配置 file_integrity 模块

```
auditbeat.modules:
-module: file_integrity
  paths:
  - /bin
  - /usr/bin
  - /sbin
  - /usr/sbin
```

如果要让 auditbeat 将数据送入 Elasticsearch，可以在 auditbeat.yml 中添加代码段 9.29 所示的配置信息，指定 Elasticsearch 的 IP 地址和端口号。如果 Elasticsearch 已经启用了 X-Pack的 Security 机制，则需要在配置中加入身份验证信息。

```
代码段 9.29：配置 auditbeat 输出到 Elasticsearch
output.elasticsearch:
  hosts: ["localhost:9200"]              #指定 IP 地址和端口号
  username: "elastic"                    #指定用户名
  password: "elastic"                    #指定对应的密码
```

auditbeat 的索引模板 fields. yml 默认存放在/etc/auditbeat 文件夹中，在输出数据到 Elasticsearch 的功能被启用的情况下，auditbeat 启动时会自动加载默认的模板。如果要加载另一种模板，需在 auditbeat. yml 配置文件中修改相关配置，如代码段 9. 30 所示。

```
代码段 9.30：修改加载索引模板的配置信息
output.elasticsearch:
  hosts: ["localhost:9200"]
  template.name: "auditbeat"
  template.fields: "/home/cy/fields.yml"
  template.overwrite: false
```

如果 Elasticsearch 索引文件中已经存在一个模板，默认不允许覆盖已有的模板。将代码段 9. 30 最后一行的 template. overwrite：false 修改为 template. overwrite：true，即可启用覆盖功能。要手动加载模板的功能，可以在终端执行代码段 9. 31 所示的 curl 命令。

```
代码段 9.31：手动加载索引模板功能
curl -XPUT -H 'Content-Type: application/json' http://localhost:9200/
_template/auditbeat-6.2.4 -d @auditbeat.template.json
```

代码段 9. 32 所示的命令可以用来删除旧数据，并以此来强制 Kibana 在可视化过程中使用新数据。

```
代码段 9.32：删除索引文件中的旧数据
curl -XDELETE 'http://localhost:9200/auditbeat-*'
```

9.5.3　启动和关闭

在终端中使用如下命令来启动 auditbeat：

```
sudo service auditbeatstart
```

启动后，终端界面将会输出如图 9. 25 所示的日志信息。

```
cy@cy-N53SN:~$ sudo service auditbeat start
[sudo] cy 的密码:
cy@cy-N53SN:~$
```

图 9.25　启动 auditbeat

auditbeat 启动后，即开始采集服务器端操作系统或服务程序的相应数据指标并传输到 Elasticsearch 中。此时可以执行代码段 9.33 所示的 curl 命令，验证服务器端的统计数据是否已被传输到 Elasticsearch 中。

代码段 9.33：测试 auditbeat 的采集和传输功能

```
curl -H 'Content-Type: application/json' -XGET 'http://localhost:9200/
auditbeat-*/_search?pretty'
```

如需关闭 auditbeat，可以在终端执行如下命令：

```
sudo service auditbeat stop
```

此时终端界面将输出如图 9.26 所示的日志信息。

```
cy@cy-N53SN:~$ sudo service auditbeat stop
cy@cy-N53SN:~$
```

图 9.26　关闭 auditbeat

9.5.4　使用 Kibana 进行展示

在 Kibana 的 Management 组件中添加一个新的索引模式，在过滤索引文件名称的输入框中填写 auditbeat-* 来匹配所有由 auditbeat 创建的索引，然后在下面的 Time-field name 中选择@timestamp，单击 Create index pattern 按钮，即可添加 auditbeat 相关索引模式，如图 9.27 所示。

使用条形统计图、统计数值、饼图和表格等不同的可视化统计图表，对索引文档中的数据指标总量、时间分布、事件序列号、文件大小等统计信息进行可视化展示，并将所有可视化结果添加到一个动态仪表板中。图 9.28 中的信息表明了各类系统进程相关指标数据在总体中的统计量和比率，其中，左上方的统计数据表示在索引中存储的进程指标数据按时间分布的情况，右上方的统计数值显示了索引中已采集到的数据总量，左下方的饼图展示了在当前查看的时间段内由系统分配的事件序列号数据比率；右下方的表格则对不同文件尺寸数据指标进行了统计。

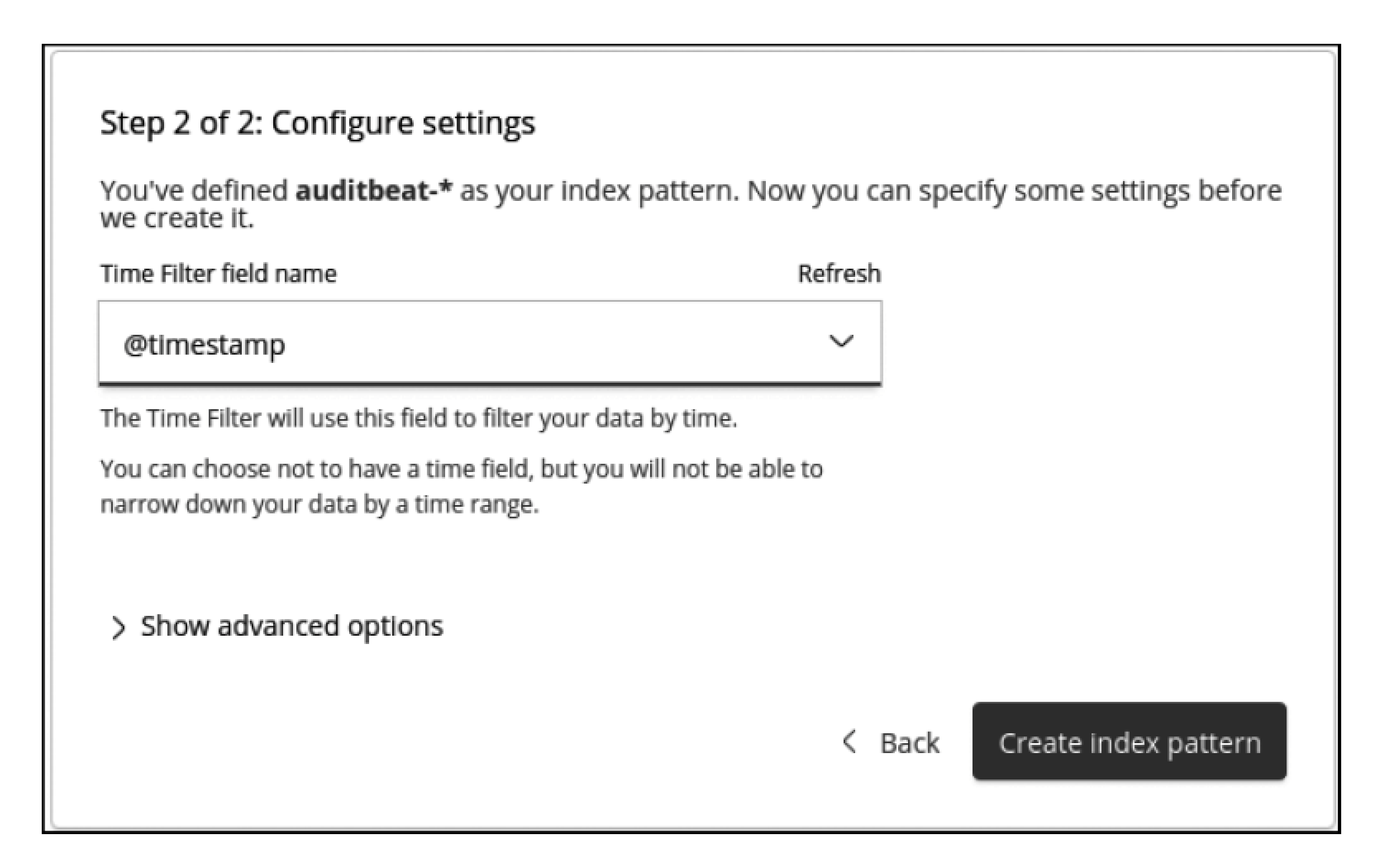


图 9.27　添加索引模式

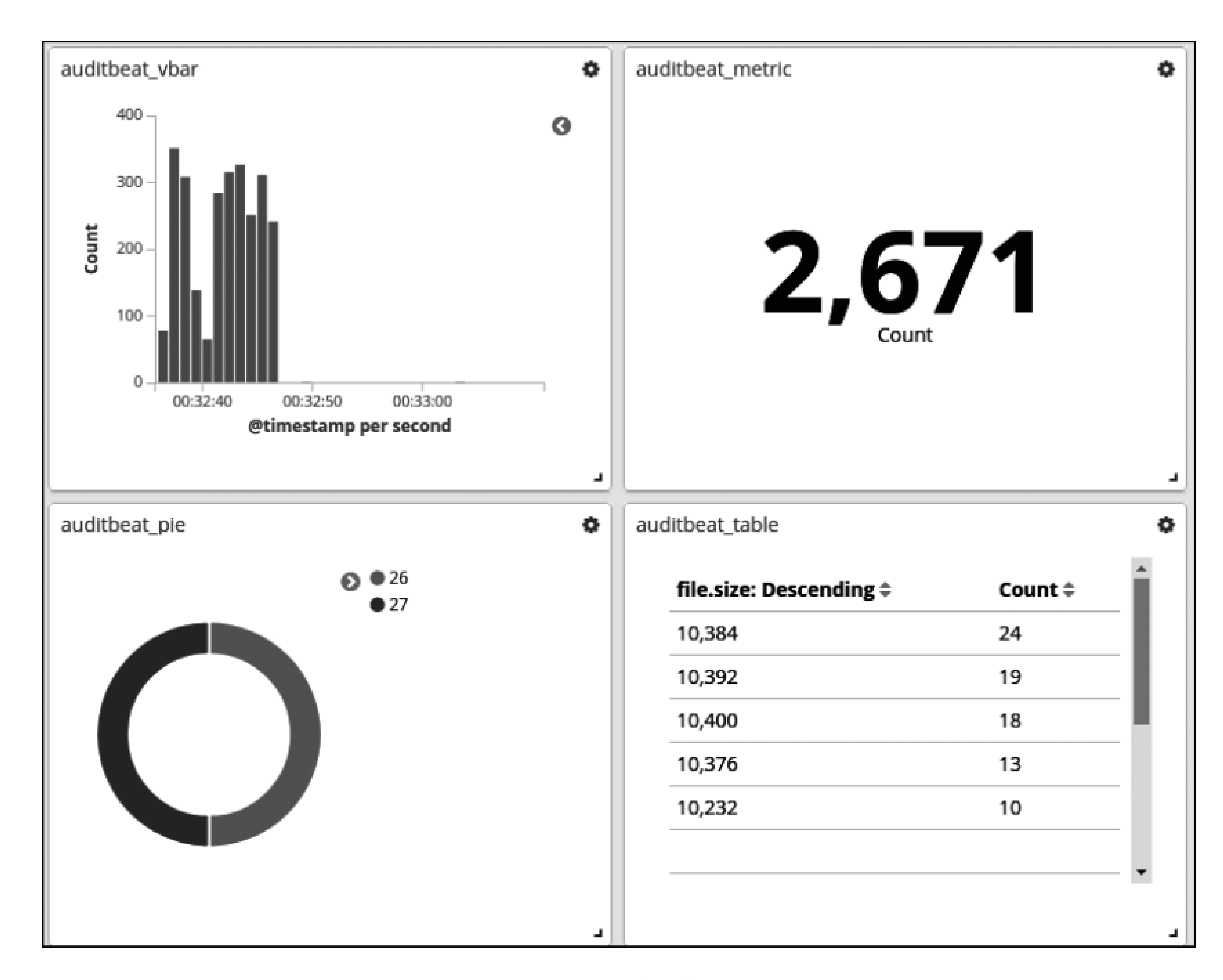


图 9.28　可视化展示

9.6 基于 heartbeat 的服务状态检测

9.6.1 概述

heartbeat 是一个轻量级守护程序，可以安装在远程服务器上，定期检查服务状态并确定其是否可用。与 metricbeat 仅可以表明服务器启停的特性不同，heartbeat 会体现出服务是否可访问。在应用场景上，heartbeat 可以验证服务正常运行的服务级别协议，验证外部无法访问私有企业服务器上的服务，以及检查所有负载平衡的服务是否可用，等等。heartbeat 以监视器为单位运行，每个监视器都会根据配置文件指定的计划运行。

9.6.2 安装和配置

安装 heartbeat 之前，需要确保 Elastic Stack 相关软件产品 Elasticsearch、Logstash（可选）和 Kibana 已经完成安装和配置。在终端运行下面第一个命令，从 Elastic Stack 官网获取 heartbeat 的 DEB 安装包（其中的 6.2.4 是版本号）。获取安装包之后，在终端执行第二行命令，完成解压并安装 heartbeat。

```
curl -L -O https://artifacts.elastic.co/downloads/beats/heartbeat/heartbeat-
6.2.4-amd64.deb
sudo dpkg -i heartbeat-6.2.4-amd64.deb
```

程序默认保存在/etc/init.d 目录下，其配置文件 heartbeat.yml 默认保存在/etc/heartbeat 目录下。在 heartbeat.yml 配置文件中可以对监视器（monitors）模块进行配置来确认一台远程主机的运行状态。对于不同的主机，可以添加不同的配置。代码段 9.34 演示了分别以 ICMP 和 TCP 方式来监视远程主机运行状态的配置。其中的 schedule 参数指定了访问的频率，mode 参数指定了 IP 解析的方式。

代码段 9.34：为 Heartbeat 配置 monitors 模块

```
heartbeat.monitors:
- type: icmp
  schedule: '*/1 * * * * * *'          #每 1min 一次
  hosts: ["myhost"]
- type: tcp
  schedule: '@every 5s'                #每 5s 一次，这里使用了@every 关键字
  hosts: ["myhost:12345"]
  mode: any                            #只需解析到任一可用 IP 地址即可
```

上述配置信息中的 schedule 参数使用了 cronexpr 表达式（其全称为 golang cron expression parser，该表达式包含 7 个字段，以空格隔开，分别为秒、分、时、日、月、周、年，其中的分、时、日、月、周为必选字段）来对 cron 表达式进行解析，用来表达某种任务执行的频率。cronexpr 各字段的详细说明如表 9.1 所示。

表 9.1　cronexpr 各字段详细说明

字段名称	是否必选	有效范围取值	有效特殊字符
秒	否	0～59	* /, -
分	是	0～59	* /, -
时	是	0～23	* /, -
日	是	1～31	* /, - L W
月	是	1～12 或 JAN～DEC	* /, -
周	是	0～6 或 SUN～SAT	* /, - L #
年	否	1970～2099	* /, -

表格最后一列所述特殊字符的含义如下：

- 星号（*）：表示 cron 表达式匹配字段的所有值。例如，在第 4 个必选字段（月）中使用星号表示每个月。
- 斜线（/）：描述范围的增量。例如，分字段中的 3-59/15 表示每个小时的第 3 分钟之后每 15min。
- 逗号（,）：用于分隔同一字段中的多个项目。例如，在第 5 个必选字段（周）中使用“MON，WED，FRI”表示星期一、星期三和星期五。
- 连字符（-）：定义范围。例如，2016-2018 表明年份从 2016 年至 2018 年，包含边界值。
- L：代表最后（last）。当在周字段中使用时，它允许用户指定诸如“上周五”（5L）之类的条件。在日期字段中，它指定该月的最后一天。
- W：用于日期字段，指定最接近给定日期的工作日（周一至周五）。例如，如果指定 15W 作为日期字段的值，则含义为距离当月 15 日最近的工作日。如果 15 日是星期六，触发器将在 14 日（星期五）被触发；如果 15 日是星期日，触发器将在 16 日（星期一）被触发；如果 15 日是星期二，那么它将在 15 日当天被触发；但是如果指定 1W，并且该月 1 日为星期六，则触发器将在 3 日（星期一）被触发，因为它不会跨越不同月份的边界。需要注意的是，当且仅当指定的日期为单独的一天（而不是一个连续的日期范围或某些特定日期的列表）时，才能使用 W 字符。它也可以与 L 组合，即 LW 表示该月的最后一个工作日。

- 哈希（#）：用于构建表示“某月的第几周的星期几”的表达式，# 后面必须跟数字 1～5。例如，5#2 表示某月的第二个星期五。

如果让 heartbeat 将数据送入 Elasticsearch，可以在 heartbeat.yml 中添加代码段 9.35 所示的配置信息，指定 Elasticsearch 的 IP 地址和端口号。如果 Elasticsearch 已经启用 X-Pack 的 Security 机制，则需要在配置中加入身份验证信息。

代码段 9.35：配置 heartbeat 输出到 Elasticsearch

```
output.elasticsearch:
  hosts: ["localhost:9200"]                 #指定 IP 地址和端口号
username: "elastic"                         #指定用户名
  password: "elastic"                       #指定对应的密码
```

heartbeat 的索引模板 fields.yml 默认存放在/etc/heartbeat 文件夹中，在输出数据到 Elasticsearch 的功能被启用的情况下，heartbeat 启动时会自动加载默认的模板。如果要加载另一种模板，需在 heartbeat.yml 配置文件中修改相关配置，如代码段 9.36 所示。

代码段 9.36：修改加载索引模板的配置信息

```
output.elasticsearch:
  hosts: ["localhost:9200"]
  template.name: "heartbeat"
  template.fields: "/home/cy/fields.yml"
  template.overwrite: false
```

如果 Elasticsearch 索引中已经存在一个模板，默认不允许覆盖已有的模板。将代码段 9.36 最后一行的 template.overwrite: false 修改为 template.overwrite: true，即可启用覆盖功能。要手动加载模板的功能，可以在终端执行代码段 9.37 所示的 curl 命令。

代码段 9.37：手动加载索引模板功能

```
curl -H 'Content-Type: application/json' -XPUT 'http://localhost:9200/
_template/heartbeat -d @ /etc/heartbeat/heartbeat.template.json
```

代码段 9.38 所示的命令可以用来删除旧数据，并以此来强制 Kibana 在可视化过程中使用新数据。

代码段 9.38：删除索引文件中的旧数据

```
curl -XDELETE 'http://localhost:9200/heartbeat-* '
```

9.6.3　启动和关闭

在终端中使用如下命令来启动 heartbeat：

```
sudo service heartbeat-elastic start
```

终端界面将会输出如图 9.29 所示的日志信息。

```
cy@cy-N53SN:~$ sudo service heartbeat-elastic start
[sudo] cy 的密码:
cy@cy-N53SN:~$
```

图 9.29　启动 heartbeat

heartbeat 启动后，即开始采集服务器端操作系统或服务程序的相应数据指标并传输到 Elasticsearch 中，此时可以执行代码段 9.39 所示的 curl 命令，验证服务器端的统计数据是否已被传输到 Elasticsearch 中。

代码段 9.39：测试 heartbeat 的采集和传输功能

```
curl -H 'Content-Type: application/json' -XGET 'http://localhost:9200/
heartbeat-*/_search?pretty'
```

如需关闭 heartbeat，可以在终端执行如下命令：

```
sudo service heartbeat-elastic stop
```

此时终端界面将输出如图 9.30 所示的日志信息。

```
cy@cy-N53SN:~$ sudo service heartbeat-elastic stop
[sudo] cy 的密码:
cy@cy-N53SN:~$
```

图 9.30　关闭 heartbeat

9.6.4　使用 Kibana 进行展示

在 Kibana 的 Management 组件中添加一个新的索引模式，在过滤索引文件名称的输入框中填写 heartbeat-* 来匹配所有由 heartbeat 创建的索引文件，然后在下面的 Time-field name 中选择@timestamp，单击 Create 按钮即可添加 heartbeat 的相关索引模式，如图 9.31 所示。

使用条形统计图、热力图、饼图和统计数值等不同的可视化统计图表，对索引文件中的数据指标总量、传输时延统计、访问方式、各种方式访问数量等统计信息进行可视化展示，并将所有可视化结果添加到一个动态仪表板中。图 9.32 中的信息表明了各类系统运行指标

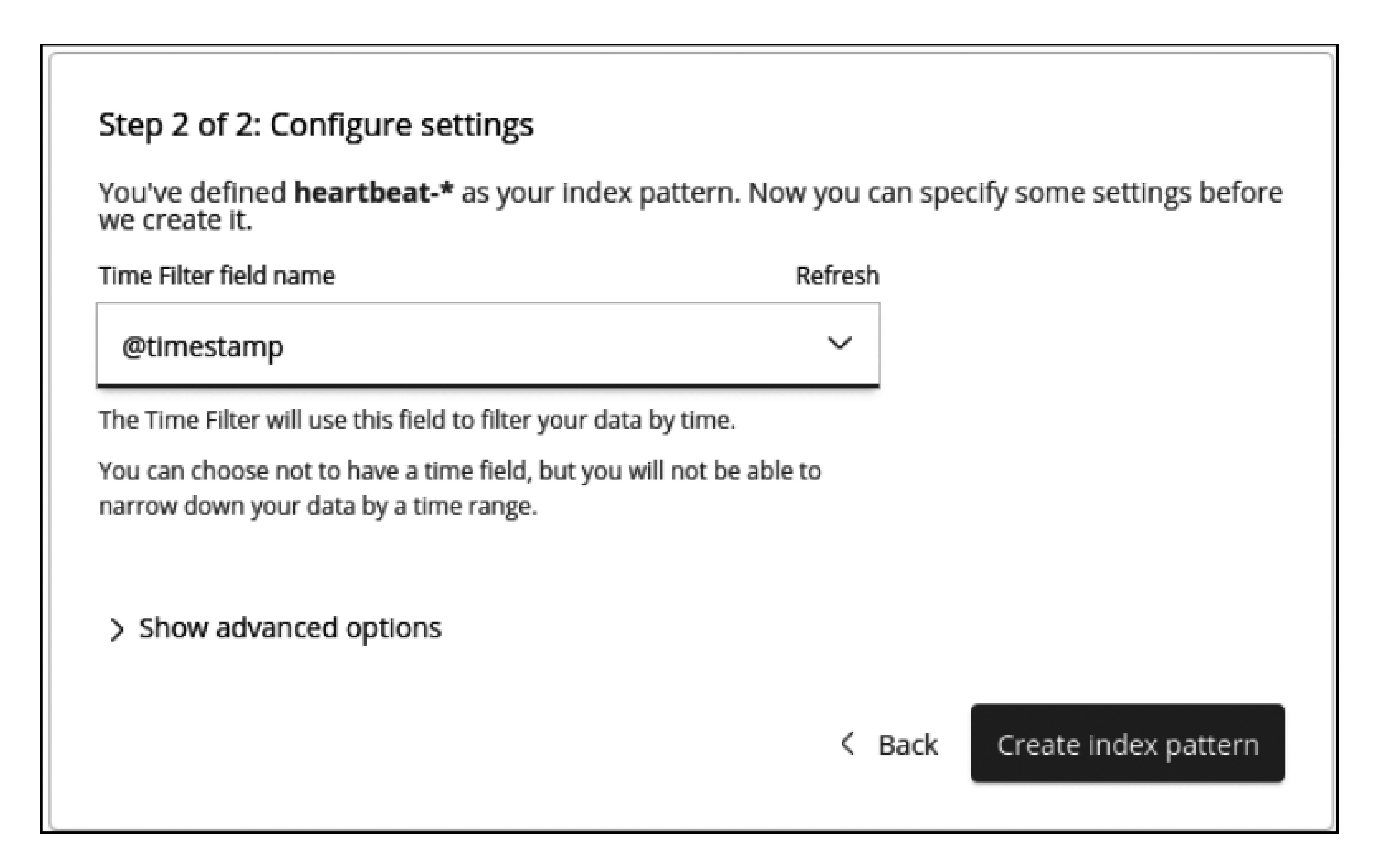


图 9.31　添加索引模式

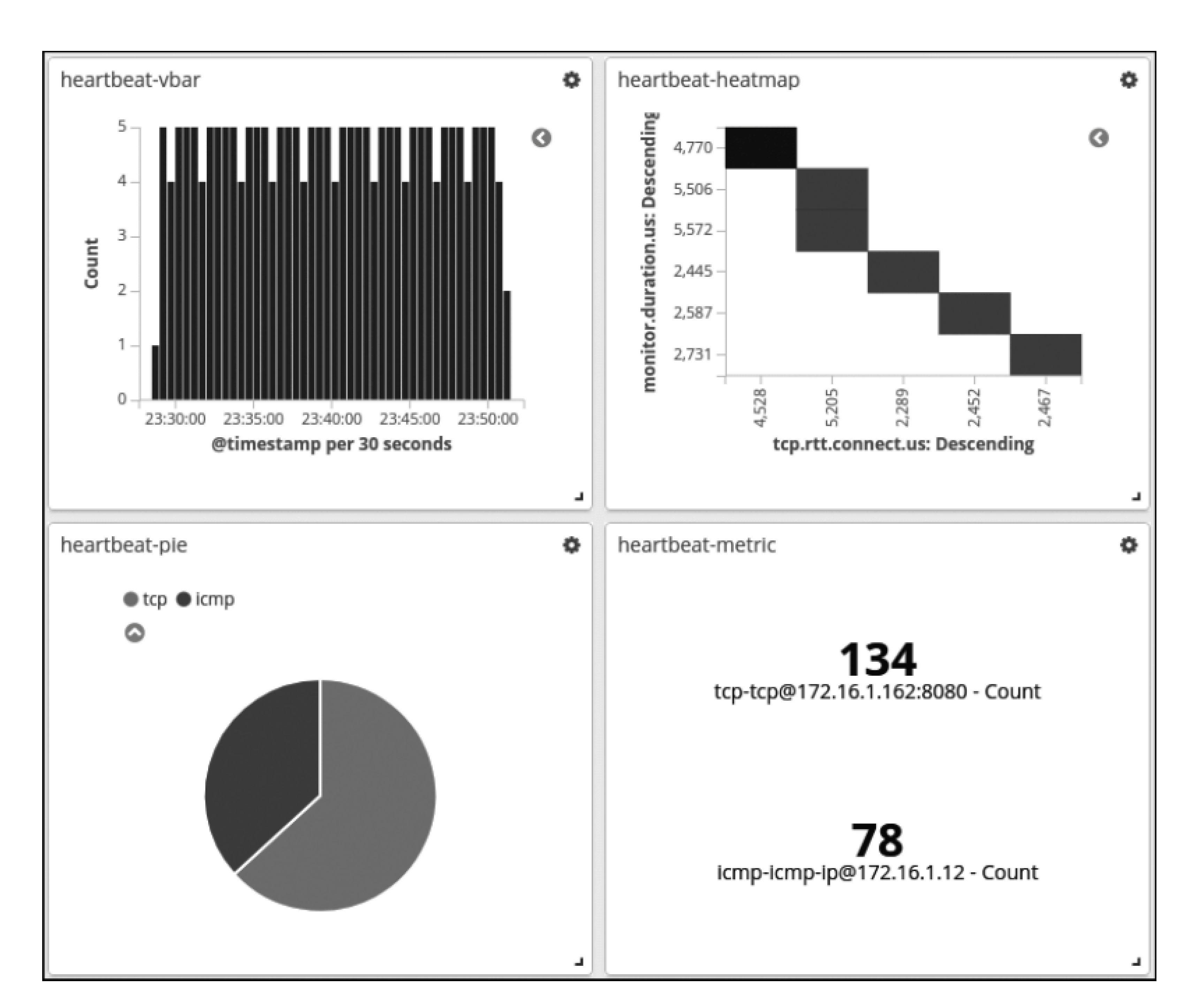


图 9.32　可视化展示

数据在总体中的统计量和比率，其中，左上方的条形图展示了远程主机状态获取量按时间分布情况，右上方的热力图表示两种传输时延的相对数量关系，左下方的饼图反映了两种不同

方式网络请求的比率,右下方的数值统计出了两种方式请求的具体数量。

9.7　扩展知识与阅读

系统运行数据的内容从侧面反映了服务器在运行时软件层面上的基本状况。对运行数据的查询和分析是运维的一个重要组成部分。当前众多对外提供服务的应用系统均对其软件层面实施各种优化和均衡策略,旨在将程序以最轻量且高效的形式运行,同时平衡软硬件系统中各部分的负载,以防止出现性能方面的故障(甚至服务器不堪重负,出现宕机等严重问题),因此,对运行数据进行监控就显得非常必要。ElasticStack 官方针对这方面提出的解决方案便是 Beats,其显著优点在于成体系和全自动。Beats 中各类数据传输工具全部基于 libbeat 平台,与 Elastic Stack 系列产品无缝对接。此外,Beats 工具一经安装配置,可以在完全无人值守的条件下执行,直接存储相关数据,使得用户可以轻松掌握软件层次的各种数据指标,及时应对软件方面的问题。

9.8　本章小结

本章对 Beats 组件中的 6 种通用数据传输工具的安装、配置、使用和可视化方法进行了介绍。通过对本章内容的了解和实践,可以实现对服务器端网络数据包收发情况、日志数据信息、系统运行状态和指标、Windows 系统中事件日志信息、用户和进程活动信息、远程主机运行状态信息等数据的采集、解析和传输。通过 Elasticsearch 分布式搜索引擎进行存储和检索,并利用 Kibana 进行各类数据的可视化展示。Beats 的应用可以使用户对系统中正在吞吐的各种数据信息有宏观上的把握,这在使用 X-Pack 进行应用系统层面监控的同时,提供了另一种不同的监控视角,有利于用户及时从数据层面了解当前系统性能演变趋势,更加有效而全面地实现对系统整体数据指标的评价和分析。

信息检索与分析实例(一)

> Get insight into how various organizations are using elastic stack products to tackle a growing number of use cases. Symantec：successfully switched from solr to Elasticsearch with Elastic Support；USAA：securing USAA's entire internal network and application portfolio(continued).
>
> https://www.elastic.co/use-cases

随着大数据、大型综合网站以及 Web 2.0 技术的普及,越来越多的软件开发者需处理海量异构信息的索引、检索、日志挖掘、可视化等和信息检索与大数据挖掘相关的业务。本章给出使用 Elasticsearch、Logstash、Kibana 针对动态网页内容的信息检索与应用实例。结合某行业需求,给出如何对行业案例进行存储优化的方案(后台使用了 Java Web 框架 Spring Boot,前端使用 Layui 和 jQuery 框架,通过 Ajax 进行前后端数据交互),对 Web 检索端的设计和后台日志输出、日志和行业案例的可视化分析等内容进行叙述,给出针对相似案例和规章制度的检索与分析。

10.1 基于 Elasticsearch 的行业信息存储

Elasticsearch 可以以 JSON 格式存储数据,每个字段类型都有相应的规则。存储数据时,需要按照对应的字段类型存储,以方便后期维护和索引分析。这里所使用的数据经过了脱敏处理(后文中以机房管理规章制度为示例,相关示例文本是从网络公开内容收集来的,仅用于本章的示例分析。示例文本内容分为两个部分,一部分为故障案例,另一部分为行业的部分规章制度)。

10.1.1 环境准备

安装 Elastic Stack 的相关软件如 Elasticsearch、Logstash 和 Kibana,并为 Elasticsearch

安装中文分词器(如 IK,详见第 2 章)。在工程方面,需要事先配置好 Java 环境和 Maven 项目管理工具。

10.1.2 数据准备

行业故障案例包含两个字段:发生时间和详情内容,如图 10.1 所示。对于发生时间的存储,可使用 date 类型,并设置格式化形式为 yyyy-MM-dd;详情内容则设置为 text 类型,并设置 Analyzer 为 IK 分词器的最大匹配模式。有关索引文件的创建和映像的设置方法参见代码段 10.1。

```
2017年12月29日21时30分|**行业数据机房，UPS按照1+1设计，建设过程中，将1+1系统临时变更为两个
2017年12月29日14时00分|**数据机房，UPS 1+1架构，UPS维护时，关掉一台后，并机柜开关突然跳闸，
2017年12月26日8时30分|某**行业UPS机房电池起火。机房温度引发电池热失控，电池短路起火。
2017年12月25日5时47分|故障现象： UPS机房起火。故障过程：UPS电容故障起火，引燃UPS和UPS上方电
2017年12月21日16时47分|空调漏水引发UPS短路炸机，总结：UPS上方不容许有空调和水管等危险源的存
2017年12月20日5时22分|碳纤维加固楼板，楼板开孔引发UPS炸机。总结：动力机房使用碳纤维加固必须
2017年12月20日4时27分|UPS电池失效故障，总结：定期进行电池维护，确保电池良好。
2017年12月18日3时20分|电池连接电缆压降过大，导致电池放电终止提前。特种环境下设计必须谨慎，
2017年12月18日3时59分|某客户营业厅使用单进单出的10kVAUPS,后端负载为交换机、台式电脑、液晶屏
```

图 10.1 行业故障案例部分的数据

代码段 10.1:创建行业故障案例的索引文件和类型的映像

```
curl -H 'Content-Type: application/json' -XPUT localhost:9200/case/_mapping/
_doc -d {                                              #定义索引文件名称为 case
    "properties": {
        "content": {
            "type": "text",
            "analyzer": "ik_max_word"
        },
        "date": {
            "type": "date",
            "format": "yyyy-MM-dd"
        }
    }
}
```

行业规章制度部分的数据如图 10.2 所示。

为方便索引和定位,将每个章节和规则条目使用正则表达式进行匹配,以便分割并提取其中的章节标题等。本例中,文件内容可整理成 5 个字段,即规章制度的文件名(docName)、章节标题(chapterTitle)、章节序号(chapter)、条目序号(article)、内容(context)。对于这些字段,都将其设置为 text 类型,并设置 Analyzer 为 IK 分词器的最大匹配模式,其索引文件的创建和映像的设置方法见代码段 10.2。

```
第一章　出入管理
第一条　机房钥匙专人保管。
第二条　外单位维护人员需要进入机房工作的，需首先填写登记表，并在机房工作人员陪同下进入机房。
第三条　参观人员需事先联系，并得到中心主任批准，才能在工作人员陪同下进入机房。
第四条　严禁无关人员进入机房。
第五条　进入机房应遵守机房管理制度并听从机房工作人员指导。
第六条　进入机房不得携带任何易燃、易爆、腐蚀性、强电磁、辐射性、流体物质等对设备正常运行构成

第二章　操作管理
第七条　UPS电源是保证网络机房、数据机房内设备正常运行和数据安全的重要设备，除管理员外，未经许
第八条　管理员要了解UPS电源的工作原理，正确区分使用UPS电源供电插座和市电供电插座，不得在UPS电
第九条　定期对UPS电源充放电。当市电不停时，应每3个月对UPS电源的电池组进行一次维护性放电。
第十条　在值班期间，值班人员发现UPS电源有异常情况要及时报告管理员，并采取适当应急处理措施。
```

图 10.2　行业规章制度部分的数据

代码段 10.2：创建行业规章制度的索引文件和类型的映像

```
curl -H 'Content-Type: application/json' -XPUT localhost:9200/doc/_mapping/rule -d {
                                                          #定义索引文件名称为 doc
    "properties": {
        "docName": {
            "type": "text",
            "analyzer": "ik_max_word",
            "search_analyzer": "ik_max_word",
            "boost": 2
        },
        "chapterTitle": {
            "type": "text",
            "analyzer": "ik_max_word",
            "search_analyzer": "ik_max_word"
        },
        "chapter": {
            "type": "text",
            "analyzer": "ik_max_word",
            "search_analyzer": "ik_max_word"
        },
        "article": {
            "type": "text",
            "analyzer": "ik_max_word",
            "search_analyzer": "ik_max_word"
        },
        "context": {
            "type": "text",
```

```
                "analyzer": "ik_max_word",
                "search_analyzer": "ik_max_word"
            }
        }
```

10.2 基于 Spring Boot 的信息检索及 Web 端设计

本节介绍信息检索及 Web 端设计。后台框架使用 Spring Boot,前端使用 layui 框架和 jQuery。

10.2.1 创建和配置工程

创建项目,在弹出的新建项目窗口中选中 Spring Initializr 并单击下一步,如图 10.3 所示。按要求填写工程的基本信息,即可自动生成 Spring Boot 的工程结构。需要注意的是,在第三步选择依赖时须勾选 Web,如图 10.4 所示,其他选项可视情况而定。

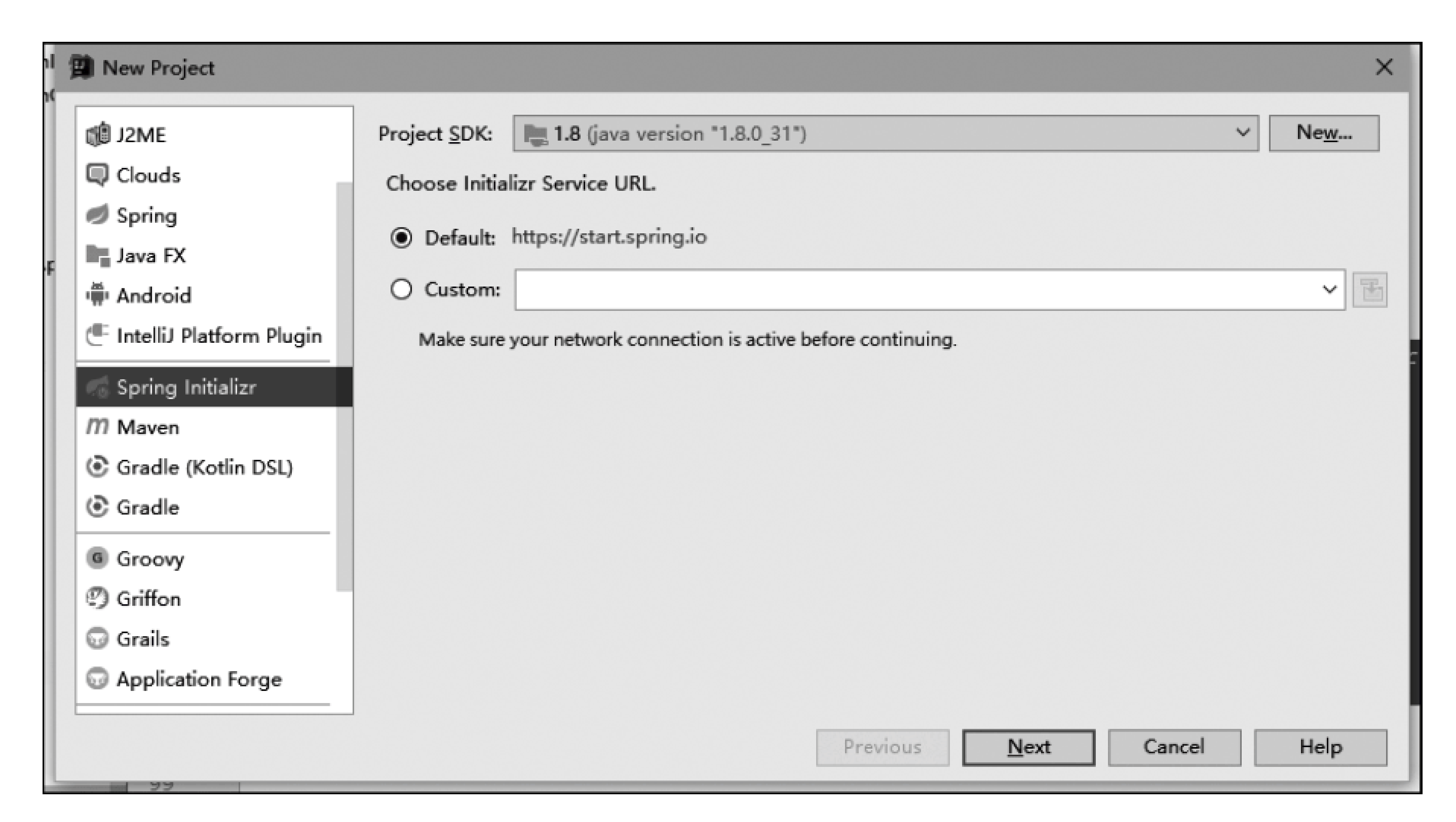


图 10.3 新建项目窗口

由于使用 Maven 构建工程,所以在导入其他依赖 JAR 包时,也选择使用 Maven 进行导入。工程还需要导入其他一些第三方 JAR 包。这里使用的是 Elasticsearch 6.2.3 的 Java 客户端(具体版本号可根据实际情况更改)以及 Alibaba 推出的 fastjson 工具。视情况需要修改 pom.xml 文件,如代码段 10.3 所示。

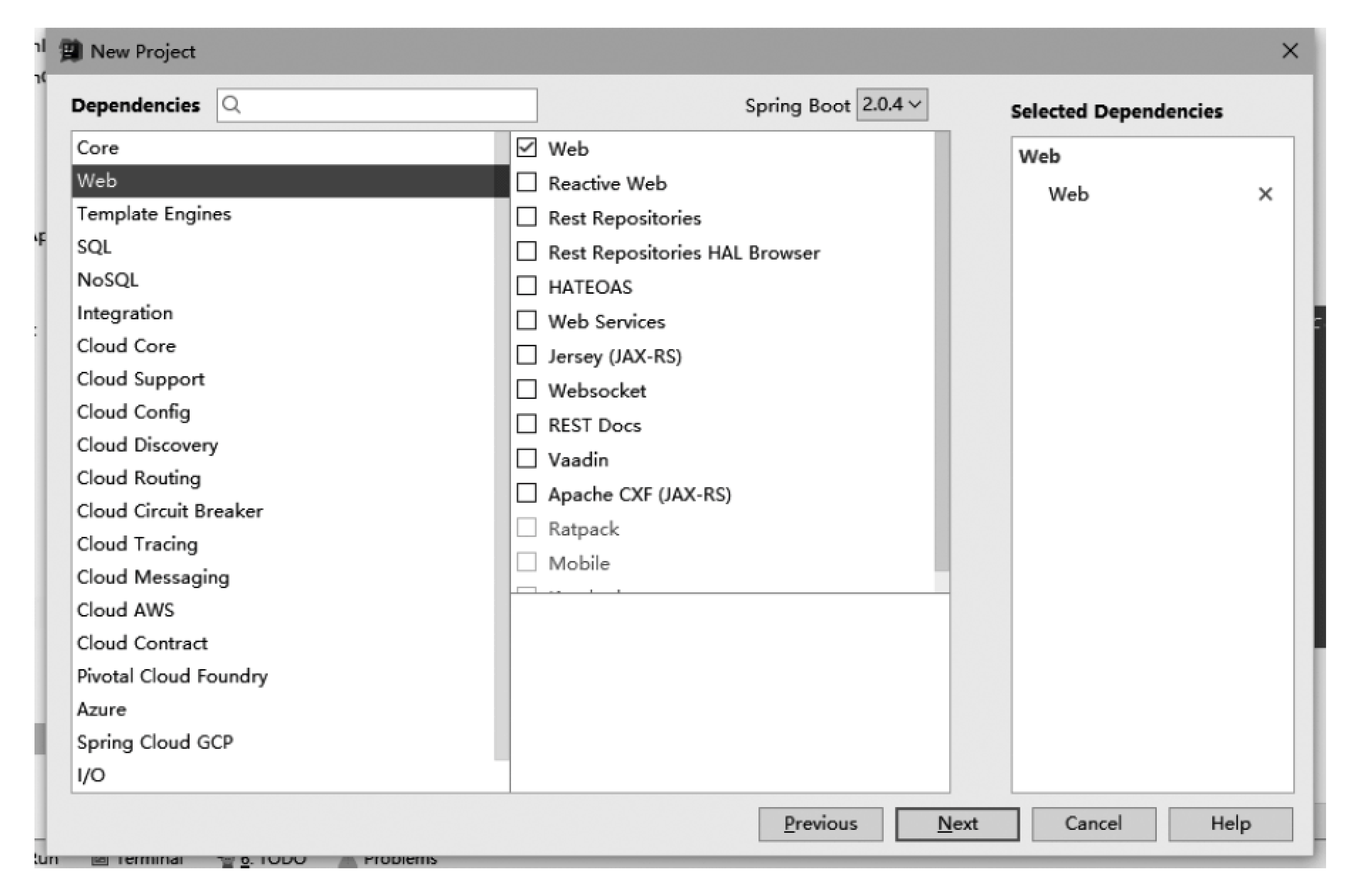


图 10.4　项目依赖选择

代码段 10.3：使用 Maven 构建工程所需 JAR 包

```
<!-- ES -->
<dependency>
  <groupId>org.elasticsearch.client</groupId>
  <artifactId>transport</artifactId>
  <version>6.2.3</version>
</dependency>
<dependency>
  <groupId>org.elasticsearch</groupId>
  <artifactId>elasticsearch</artifactId>
  <version>6.2.3</version>
</dependency>
<dependency>
  <groupId>org.elasticsearch.plugin</groupId>
  <artifactId>transport-netty4-client</artifactId>
  <version>6.2.3</version>
</dependency>
<!-- fastjson -->
<dependency>
```

```
    <groupId>com.alibaba</groupId>
    <artifactId>fastjson</artifactId>
    <version>1.2.47</version>
</dependency>
```

pom.xml 编辑好后，IDE 环境（如 IntelliJ IDEA）会自动下载新添加的 JAR 包。若没有自动导入，则需要在 IntelliJ IDEA 中打开 pom.xml，右击该文件，在快捷菜单中选择 Maven→Reimport 命令，可重新导入工程所需 JAR 包。

> Tips 使用 Spring Boot 最大的优点是通过较少的配置即可实现丰富的功能。它内置 Tomcat 服务器，运行时只需将工程导出为 JAR 包，直接运行即可。

配置上述文件后，整个项目的文件结构如图 10.5 所示。

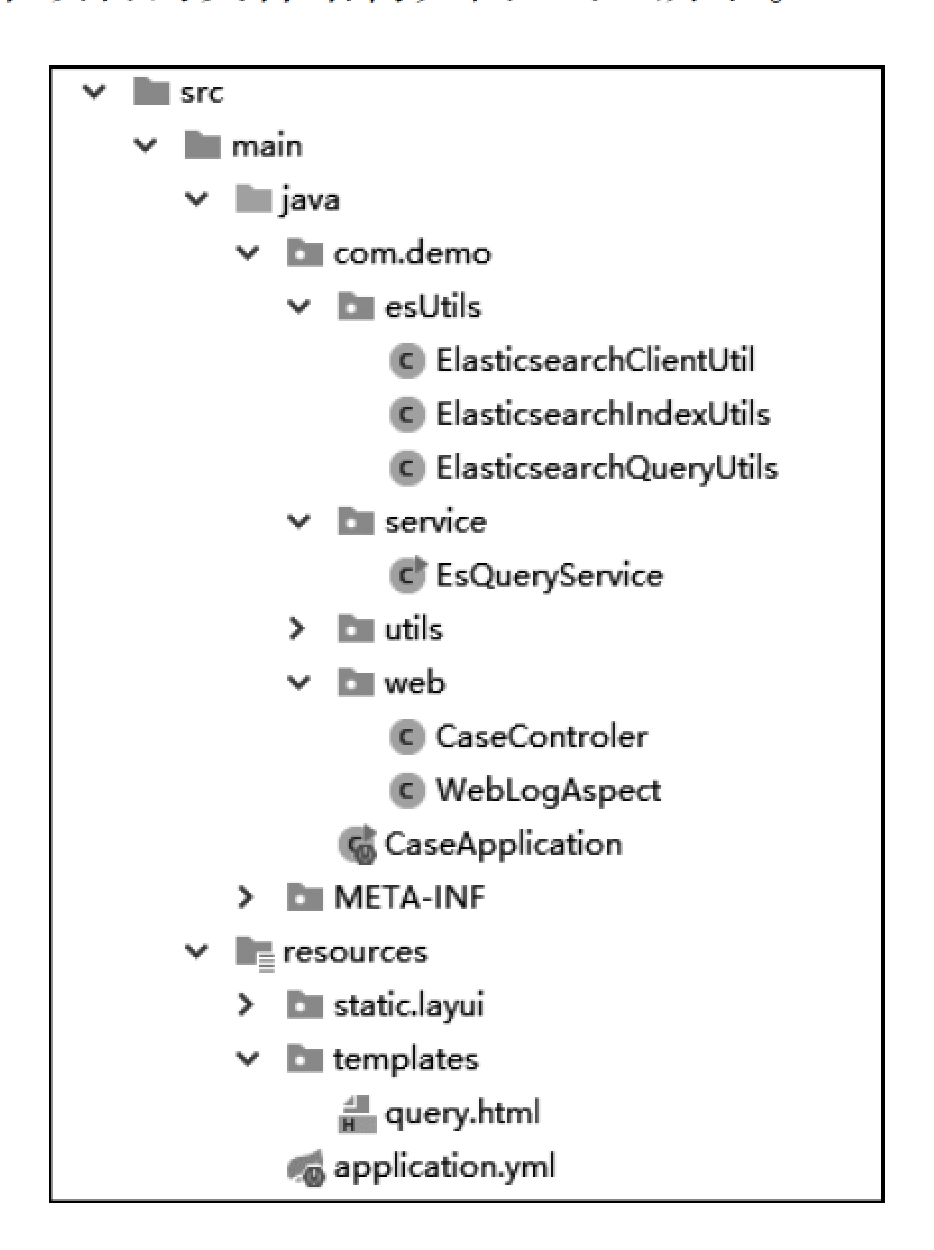


图 10.5　项目的文件结构

10.2.2　Web 页面设计

在 src/main/java/com/demo 目录下创建 Elasticsearch 的工具类，它主要用于创建客户端以及执行查询语句（见代码段 10.4）。为简单起见，客户端的创建使用单例模式。

代码段 10.4：Elasticsearch 工具类

```
import org.elasticsearch.action.search.SearchResponse;
import org.elasticsearch.action.search.SearchType;
import org.elasticsearch.client.transport.TransportClient;
import org.elasticsearch.common.settings.Settings;
import org.elasticsearch.common.transport.TransportAddress;
import org.elasticsearch.search.SearchHit;
import org.elasticsearch.search.SearchHits;
import org.elasticsearch.index.query.QueryBuilder;
import org.elasticsearch.transport.client.PreBuiltTransportClient;
import java.net.InetAddress;
import java.net.UnknownHostException;
import java.util.Map;
public class ElasticsearchQueryUtils {
    private int size=20;
    private SearchHits hits;
    private String index;
    private String type;
    //集群名,默认值为 elasticsearch
    private static final String CLUSTER_NAME="elasticsearch";
    //ES 集群中某个节点
    private static final String HOSTNAME="localhost";
    //连接端口号
    private static final int TCP_PORT=9300;
    //构建 Settings 对象
    private static Settings settings=Settings.builder().put("cluster.name", CLUSTER_NAME).build();
    //TransportClient 对象,用于连接 ES 集群
    private static volatile TransportClient client;
    public static TransportClient getClient(){
        if(client==null){
            synchronized (TransportClient.class){
                try {
                    client=new PreBuiltTransportClient(settings)
                            .addTransportAddress(new TransportAddress(InetAddress.getByName
                            (HOSTNAME), TCP_PORT));
                } catch (UnknownHostException e) {
                    e.printStackTrace();
```

```
                    }
                }
            }
            return client;
        }
        public ElasticsearchQueryUtils() { }
        public ElasticsearchQueryUtils(String index, int size){
            this.index= index;
            this.size= size;
        }
        public ElasticsearchQueryUtils Query(QueryBuilder queryBuilder){
            SearchResponse response= ElasticsearchQueryUtils.getClient().prepareSearch(this.index)
                    .setSearchType(SearchType.DFS_QUERY_THEN_FETCH)
                    .setTypes(this.type)
                    .setQuery(queryBuilder)
                    .setSize(size)
                    .setExplain(true)
                    .get();
            this.hits= response.getHits();
            return this;
        }
        public SearchHits getHits() { return hits; }
        public int getSize() {return size; }
        public void setSize(int size) {this.size= size; }
        public void setHits(SearchHits hits) {this.hits= hits; }
        public String getIndex() {return index; }
        public void setIndex(String index) { this.index= index; }
        public String getType() { return type; }
        public void setType(String type) { this.type= type; }
    }
```

在文件夹 templates 中创建 query.html,内容如代码段 10.5 所示。这里使用的前端框架是 Layui(需要先从 Layui 的官网下载相应的 CSS 和 JS 文件并放到 resources/static 目录下,注意在 html 中填写正确的路径)。前端代码主要通过 Ajax 向后台传递查询参数,待后台在 Elasticsearch 中完成查询后,将结果返回至前端,使用 jQuery 将结果添加到页面并展示。

代码段 10.5：query.html

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8">
    <title>行业故障挖掘演示</title>
    <meta name="renderer" content="webkit">
    <meta http-equiv="X-UA-Compatible" content="IE=edge,chrome=1">
    <meta name="viewport" content="width=device-width, initial-scale=1,
    maximum-scale=1">
    <link rel="stylesheet" href="layui/css/layui.css"  media="all">
</head>
<body>
<div class="layui-container">
    <fieldset class="layui-elem-field layui-field-title" style="margin-top: 50px;">
        <legend>行业故障挖掘演示</legend>
    </fieldset>
    <div class="layui-row">
        <div class="layui-tab layui-tab-card">
            <ul class="layui-tab-title">
                <li>相似案例</li>
                <li class="layui-this">规章制度</li>
            </ul>
            <div class="layui-tab-content">
                <div class="layui-tab-item">
                    <form class="layui-form" action="">
                        <div class="layui-form-item">
                                <textarea name="caseStr" required lay-verify="required"
                                placeholder="请输入案例" class="layui-textarea"></
                                textarea>
                        </div>
                        <div class="layui-form-item">
                            <button class="layui-btn" lay-submit lay-filter="sim">立即提
                            交</button>
                            <button type="reset" class="layui-btn layui-btn-primary">重置
                            </button>
                        </div>
                        <div id="simResult"></div>
                    </form>
```

```
                    </div>
                    <div class="layui-tab-item  layui-show">
                        <form class="layui-form" action="">
                            <div class="layui-form-item">
                                <input type="text" name="queryStr" required
                                lay-verify="required" placeholder="请输入查询关键字"
                                autocomplete="off" class="layui-input">
                            </div>
                            <div class="layui-form-item">
                                <button class="layui-btn" lay-submit lay-filter="rule">立即提
                                交</button>
                                <button type="reset" class="layui-btn layui-btn-primary">重置
                                </button>
                            </div>
                            <div id="ruleResult"></div>
                        </form>
                    </div>
                </div>
            </div>
        </div>
    </div>
    <script src="layui/layui.js" charset="utf-8"></script>
    <script>
        layui.use('form', function(){
            var form=layui.form;
            var $=layui.jquery;
            form.on("submit(sim)", function (data) {
                $.ajax({
                    url:"/querySim",
                    type:"GET",
                    data:data.field,
                    success: function (msg) {
                        var data=JSON.parse(msg);
                        for(var i=0; i<data.length; i++){
                            $("#simResult").append("<blockquote class="layui-elem-quote">"+
                            data[i].content+"</blockquote>");
                        }
                        form.render(); //更新全部
                    }
                });
```

```
            return false;      //阻止表单跳转。如果需要表单跳转,作相应的修改即可
        });
        form.on("submit(rule)", function (data) {
            console.log(data);
            $.ajax({
                url:"/queryRule",
                type:"GET",
                data:data.field,
                success: function (msg) {
                    var data= JSON.parse(msg);
                    for(var i= 0; i< data.length; i+ + ){
                        $("#ruleResult").append("<blockquote class= "layui- elem- quote"> "
                            +data[i].docName+ "<br> "
                            +data[i].chapter+ data[i].article+ ": "
                            +data[i].context+ "<br> "
                            +"</blockquote> ");
                    }
                    form.render();      //更新全部
                }
            });
            return false;
        });
    });
</script>
</body>
</html>
```

在 src/main/java/com/demo 文件夹下创建 Service 层文件并将其命名为 EsQueryService，内容如代码段 10.6 所示，它主要用于对上层的 Controller 提供检索服务。

```
代码段 10.6: EsQueryService
//import 语句(略)
public class EsQueryService {
    private static ElasticsearchQueryUtils utils= new ElasticsearchQueryUtils();
    public SearchHits queryRule(String... queryStr) {
        String[] fields= {"context"};
        QueryBuilder qb= new MoreLikeThisQueryBuilder(fields, queryStr, null)
                .minTermFreq(1)
```

```
                .maxQueryTerms(12);
        utils.setIndex("doc");
        utils.setType("rule");
        return utils.Query(qb).getHits();
    }
    public SearchHits queryCase(String queryStr) {
        QueryBuilder qb= QueryBuilders.matchQuery("content", queryStr);
        utils.setIndex("case");
        utils.setType("_doc");
        return utils.Query(qb).getHits();
    }
}
```

在 src/main/java/com/demo 文件夹下创建 Controller 层文件，这里将其命名为 HighSpeedController，内容如代码段 10.7 所示。它主要用于对外提供检索接口并将查询结果拼接成 JSON 数据返回至前端。对 querySimCase 函数使用@GetMapping，设置请求路径为/querySim，使用@ResponseBody 设置返回数据转化为指定格式后写入 response 对象的 body 中，通常用来返回 JSON 数据。对 queryRule 函数使用相同的设置，请求路径设置为 queryRule。

```
代码段 10.7：HighSpeedController
//import 语句（略）
@Controller
public class HighSpeedController {
    private EsQueryService esQueryService= new EsQueryService();
    @GetMapping(value= "query")
    public String queryIndex(){
        return "query";
    }
    @ResponseBody
    @GetMapping(value= "querySim")
    public String querySimCase(@RequestParam(value= "caseStr", required= true) String caseStr){
        JSONArray jsonArray= new JSONArray();
        SearchHits hits= esQueryService.queryCase(caseStr);
        for (SearchHit hit: hits) {
            jsonArray.add(hit.getSourceAsMap());
        }
```

```
            return jsonArray.toString();
        }
        @ResponseBody
        @GetMapping(value="queryRule")
        public String queryRule(@RequestParam(value="queryStr", required=true) String queryStr){
            JSONArray jsonArray=new JSONArray();
            SearchHits hits=esQueryService.queryRule(queryStr);
            for (SearchHit hit: hits) {
                jsonArray.add(hit.getSourceAsMap());
            }
            return jsonArray.toString();
        }
    }
```

启动内置 Web 服务器，如图 10.6 所示。待启动完成后，在浏览器中输入 http://localhost:8080，可以看到检索页面首页，如图 10.7 所示。

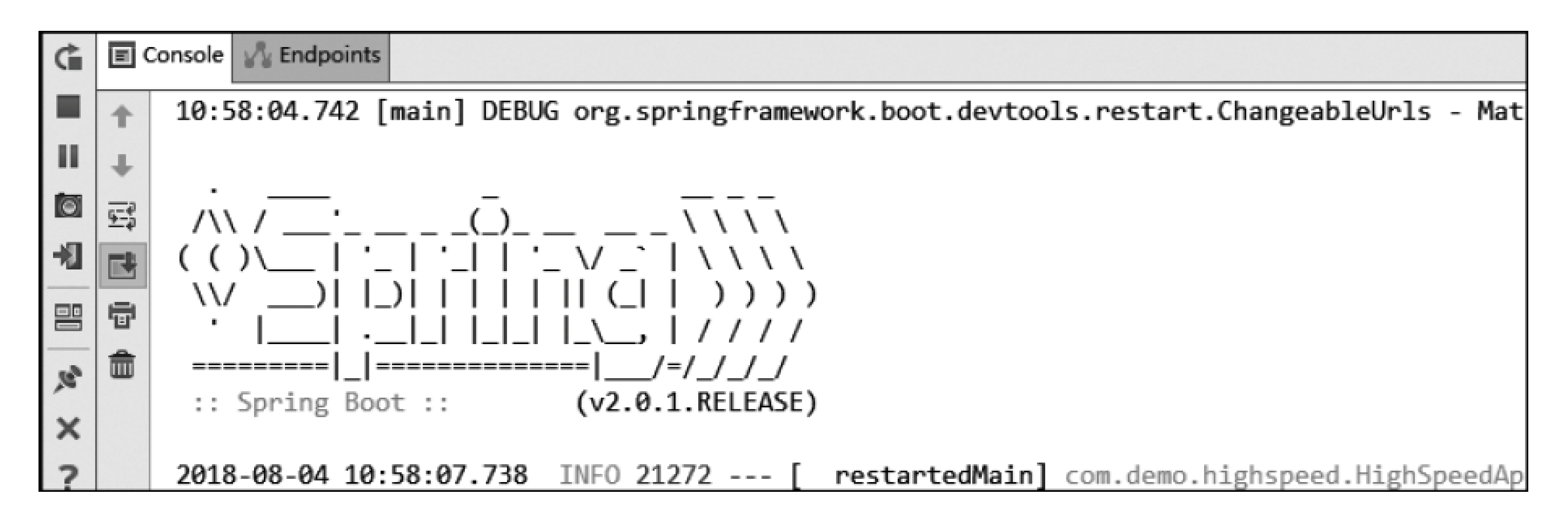


图 10.6　启动 Spring Boot 内置的 Web 服务器

行业故障挖掘演示

相似案例　规章制度

请输入案例

立即提交　重置

图 10.7　检索页面首页

在“相似案例”或“规章制度”的文本框中，输入需要查询的内容，单击“立即提交”按钮，前端会通过 Ajax 查询数据，并将查询结果集展示在前端页面。如图 10.8 和图 10.9 所示。

图 10.8　“相似案例”检索页面

图 10.9　“规章制度”检索页面

10.3　基于 Logstash 的日志处理

本节介绍基于 Logstash 的日志处理。这里将 Spring Boot 作为客户端，使用 TCP 把请求日志发送给 Logstash，输出至 Elasitcsearch 并在屏幕上输出。

10.3.1　配置 Spring Boot 输出日志

在后端和前端进行联合调试时可能会遇到很多问题。出现问题时，由于 Spring Boot 默认不输出请求日志，因而不便于查找问题。使用 AOP 解析请求，并通过 Logstash 输出至 Elasticsearch，可方便、快速地定位错误信息，同时也可为以后的维护带来便利。要使用

AOP，应先在 pom. xml 中添加依赖包的信息，如代码段 10. 8 所示。

代码段 10.8：AOP依赖包

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-aop</artifactId>
    <version>1.3.3.RELEASE</version>
</dependency>
```

Tips AOP 为 Aspect Oriented Programming 的缩写，意为面向切面编程，它是一种通过预编译方式和运行期动态代理实现程序功能的统一维护的技术。

在实现 Web 层日志切面的程序时，需要将代码段 10. 9 放到和 Controller 相同的目录下，否则无法拦截相应的请求信息。函数 doBefore 的作用是将接收到的请求信息包装成 JSON 格式的数据，并通过 TCP 传输到 Logstash 中。

代码段 10.9：转发请求

```
//import 语句（略）
@Aspect
@Component
public class WebLogAspect {
    @Pointcut("execution(* com.demo.highspeed.web.*.*(..))")
    public void weblog() {}

    @Before("weblog()")
    public void doBefore(JoinPoint joinpoint) {
        //接收到请求，记录请求内容
          ServletRequestAttributes attributes = (ServletRequestAttributes) RequestContextHolder.
          getRequestAttributes();
        HttpServletRequest request= attributes.getRequest();
        //记录请求内容
        JSONObject json= new JSONObject();
        json.put("url", request.getRequestURL().toString());
        json.put("type", request.getMethod());
        json.put("remoteIP", request.getRemoteAddr());
        json.put("method", joinpoint.getSignature().getName());
```

```
            json.put("params", Arrays.toString(joinpoint.getArgs()));

            TcpClient.sendData(json.toString());
        }
    }
```

有关 TCP 客户端传输协议的详细说明可参考第 6 章的相关内容。在查询相关案例和规章制度时有中文字符，因此需要将字符串统一成 UTF-8 编码。传输完毕后，注意要关闭该连接，见代码段 10.10。

```
代码段 10.10：TCP 客户端
//import 语句（略）
public class TcpClient {
    private static final String ip="10.8.0.3";
    private static final int port=5656;

    public static void sendData(String str){
        try {
            Socket s=new Socket(InetAddress.getByName(ip), port);
            DataOutputStream dos=new DataOutputStream(s.getOutputStream());
            dos.write(str.getBytes("UTF-8"));
            s.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
```

10.3.2　在 Logstash 中进行相关配置

在 Logstash 的配置文件（这里是在 Logstash 的 bin 文件夹下的 conf.conf 配置文件）中进行相关的配置，见代码段 10.11。同样，这里需要注意将输入时的字符编码统一为 UTF-8，与上述 TCP 传输数据时的编码相一致。

```
代码段 10.11：Logstash 的配置文件，注意，对 host=>["10.8.0.3"]要进行有针对性的修改，
如 localhost
input {
    tcp {
        host=>"0.0.0.0"          #可选项,默认为 0.0.0.0
```

```
            mode=>"server"    #取值为"server"、"client"之一,可选项,默认为"server"
            port=>"5656"      #必选项,端口号,需和通信的另一端的端口号匹配
            codec=>json{ charset=>"UTF-8" }
        }
    }
    filter {
        json {
            source=>"message"
        }
    }
    output {                  #输出目的地
        stdout{ }             #stdout 是标准输出文件
        elasticsearch {       #通过 HTTP 的方式将数据传输到 Elasticsearch 中
            hosts=>["10.8.0.3"]       #指定数组形式的 hosts 列表
        }
    }
```

启动 Elasticsearch。使用命令./logstash -f conf.conf 启动 Logstash。然后利用 Elasticsearch 的 Head 工具就可以看到 Spring Boot 的请求日志信息已经进入 Elasticsearch 的索引文件中了,如图 10.10 所示。


```
{
   "_index": "logstash-2018.08.05",
   "_type": "doc",
   "_id": "WsENCWUBQ0_WwNjnqX6R",
   "_version": 1,
   "_score": 1,
   "_source": {
      "url": "http://127.0.0.1:8080/queryRule",
      "type": "GET",
      "remoteIP": "127.0.0.1",
      "method": "queryRule",
      "params": "[继电器]",
      "@version": "1",
      "@timestamp": "2018-08-05T07:46:54.867Z",
      "host": "10.8.0.2",
      "port": 59773
   }
}
```


图 10.10　请求日志信息已被索引

10.4　基于 Kibana 的日志分析结果可视化

启动 Kibana,在左侧导航栏的 Management 中单击 Create Index Pattern 按钮创建新索引模式,如图 10.11 所示。



图 10.11 添加索引

在定义索引模式的文本框中输入 logstash-* 匹配所有的 Logstash 产生的日志文件。类似地,故障案例索引文件 case 和规章制度索引文件 doc 也需要先添加到 Kibana 中,如图 10.12 所示。

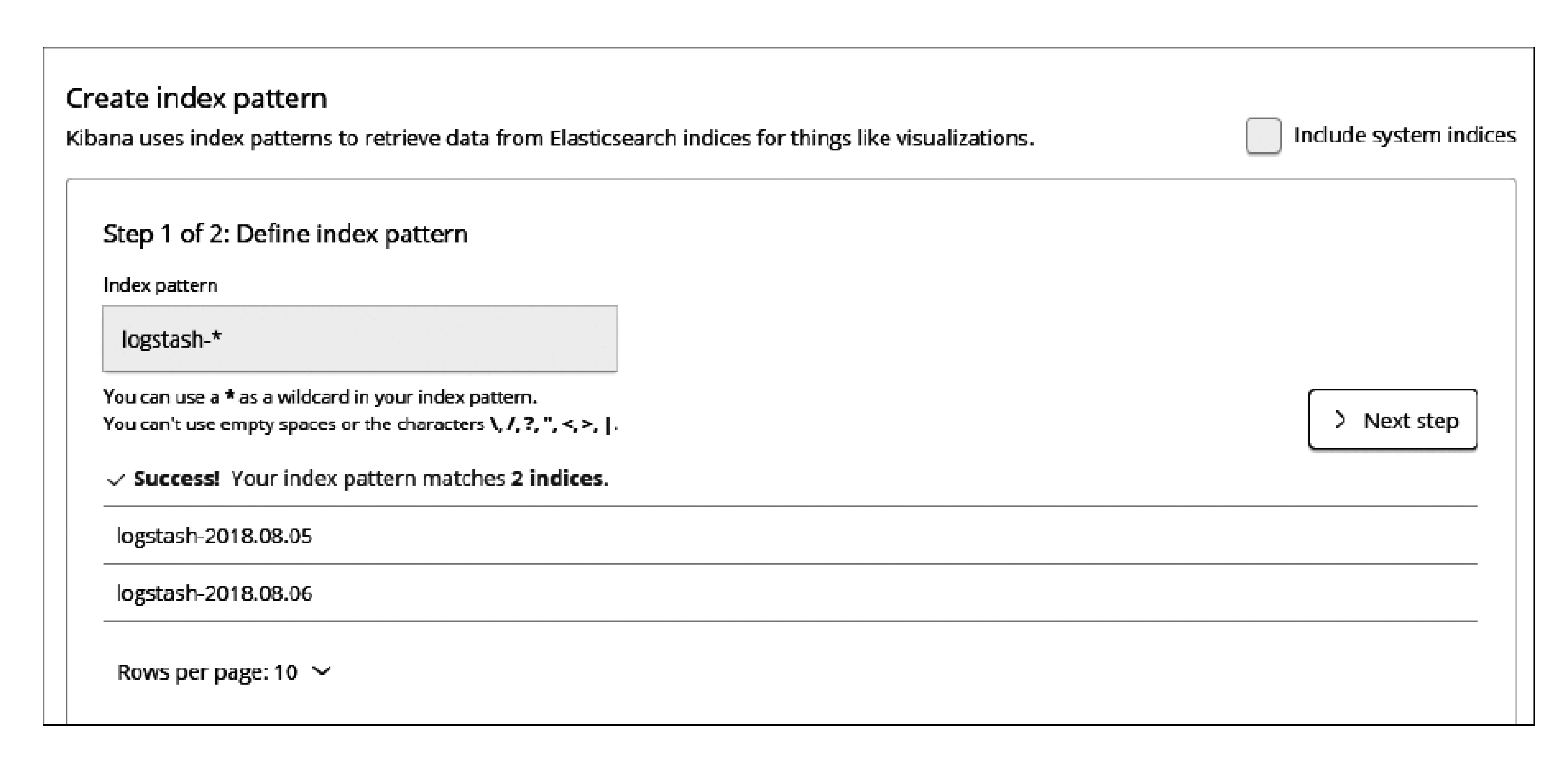


图 10.12 设置用于可视化的索引模式

10.4.1 访问量走势分析

下面对后台的请求访问量进行统计分析。在 Visualize 组件中选择 Area 选项,设置 Y 轴的聚合方式为 Count 并将自定义标签命名为"访问量",如图 10.13 所示。



图 10.13 设定图表的 Y 轴数据来源

设置 X 轴的聚合方式为 Date Histogram，将自定义标签命名为“访问时间”，其他按默认设置即可，如图 10.14 所示。

Buckets
X-Axis
Aggregation
Date Histogram
Field
@timestamp
Interval
Auto
Custom Label
访问时间
Advanced
Add sub-buckets

图 10.14　设定图表的 X 轴数据来源

单击▶按钮生成统计图，如图 10.15 所示，该图较为直观地反映了各个时间段的访问量的走势。

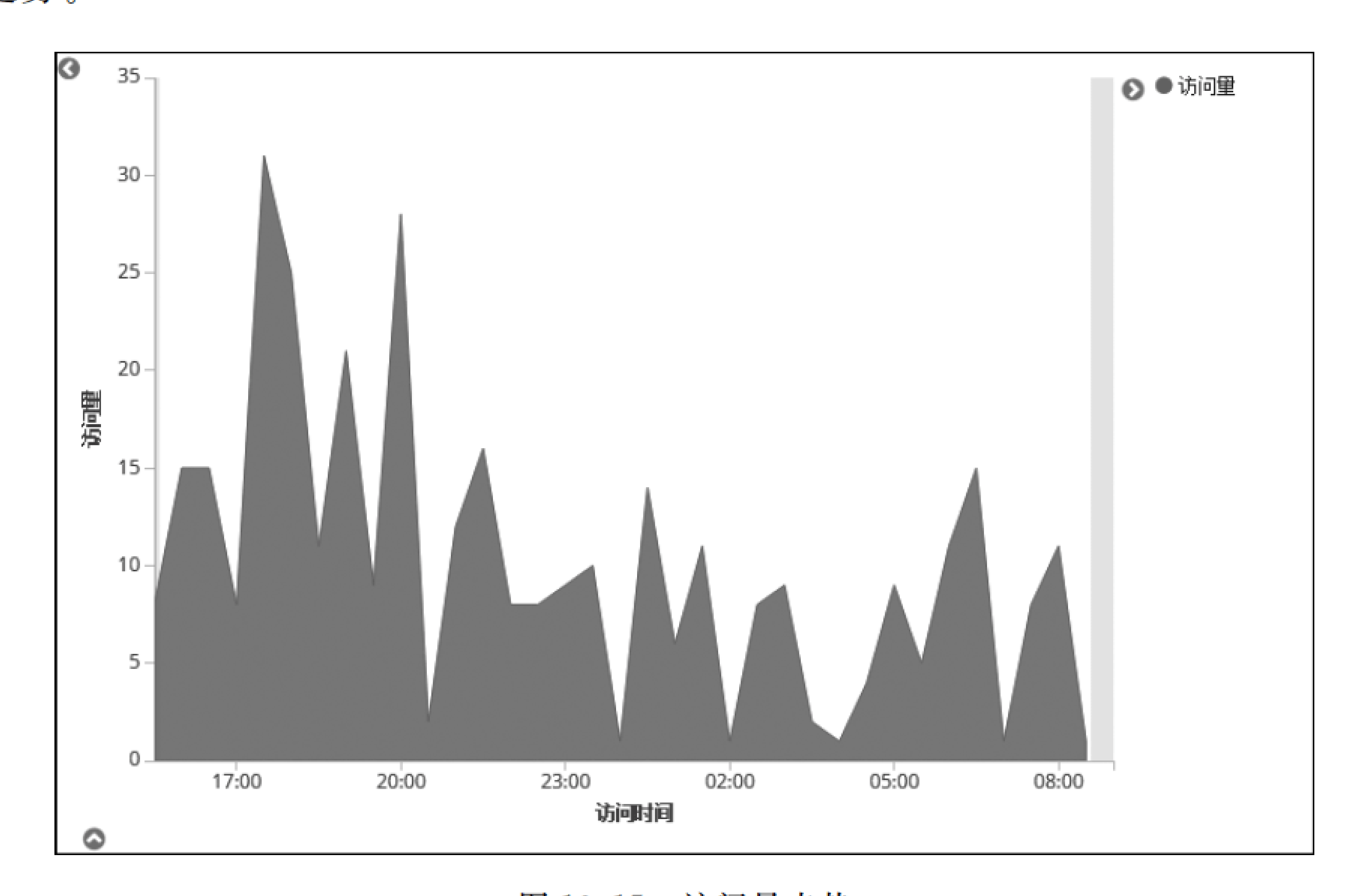


图 10.15　访问量走势

统计时，为更加精确地检索某个函数调用的次数，以便着重优化该方法，可单击上方的 Add a filter 按钮添加一个过滤器，选择需要查询的方法名称和表达式，单击 Save 按钮保存即可，如图 10.16 所示。单击☑图标可关闭或开启该过滤器。



图 10.16 添加过滤器

10.4.2 查询参数比率分析

下面给出网站请求链接中查询参数比率分析。选择 Pie 创建饼图，Bucket 的设置如图 10.17 所示，单击▶按钮生成饼图，如图 10.18 所示。借助于它，可对排名靠前的几个查询参数内容在服务器中进行缓存，对于提高服务器的响应速度有帮助。

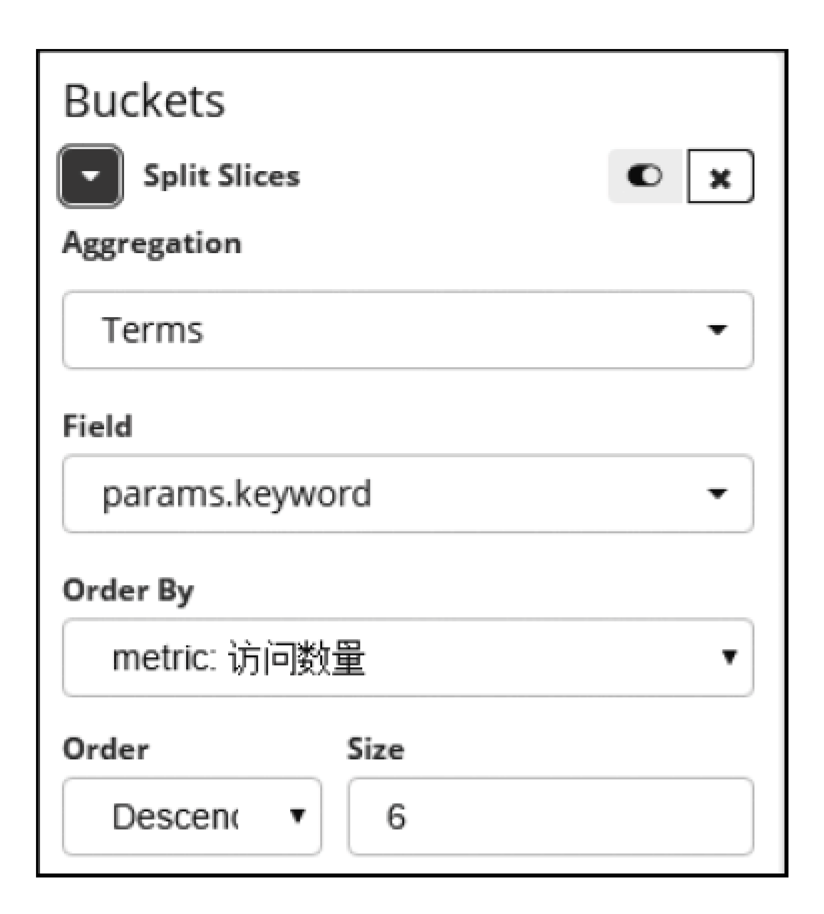


图 10.17 设定图表的统计字段

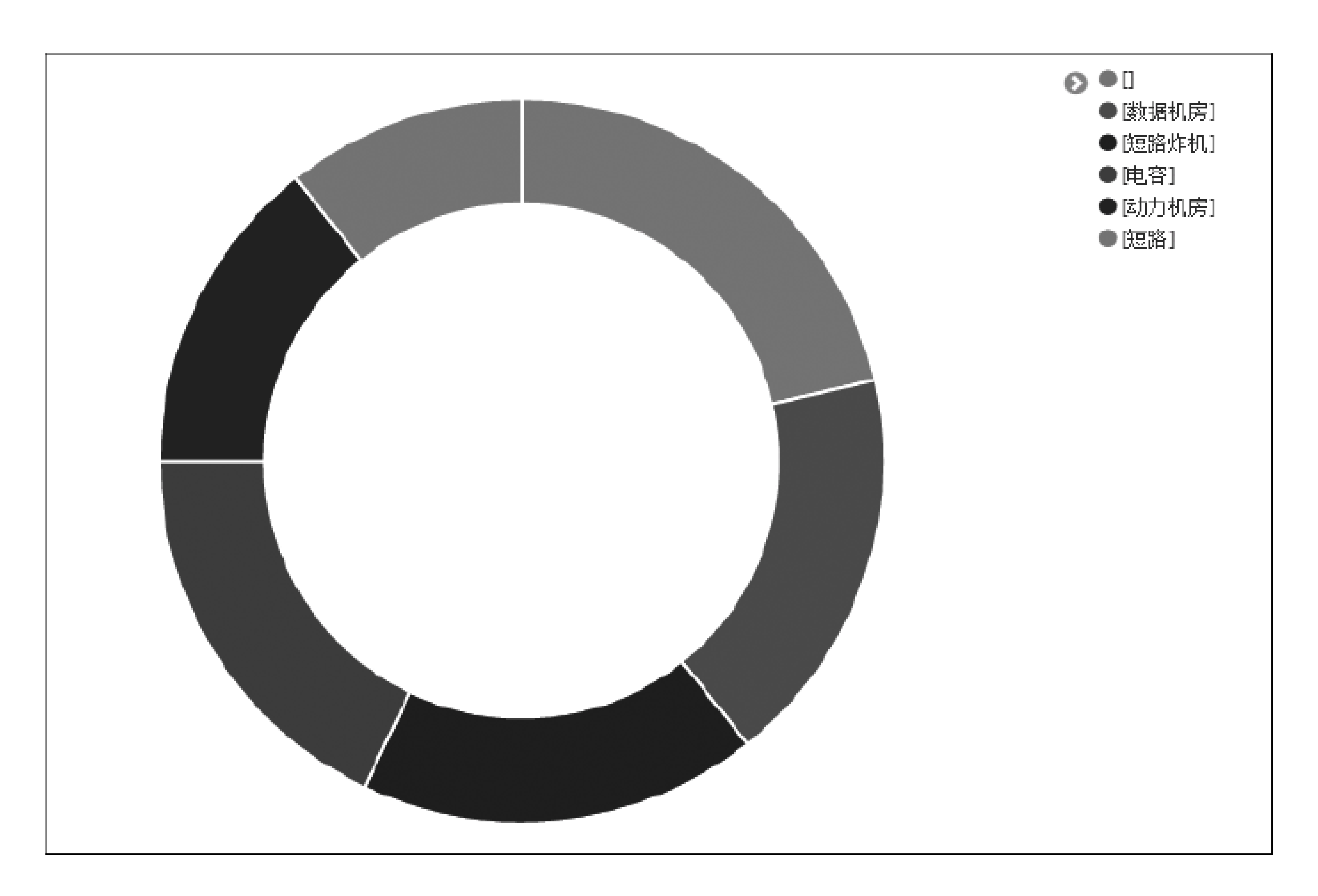


图 10.18　统计结果

10.4.3　故障案例可视化

下面给出基于 Kibana 分析在某段时间发生故障数量的方法。在 Visualize 组件中选择 Vertical Bar 选项，Y 轴设置方式可参考 10.4.2 节的内容，如图 10.19 所示。



图 10.19　设置 Y 轴聚合方式

为方便自定义统计各个时间段内发生故障的数量，可设置 X 轴的聚合方式为 Date Range，自定义各个时间段，其他按默认设置即可，如图 10.20 所示。图 10.20 中给出一种日期的计算方式，以 now-1w/d 为例，假设当前时间点为 2018 年 8 月 17 日 11:09:10，now 表示当前日期，-1w 为减去 1 周时间，/d 表示以天为单位对最后结果四舍五入，所以最后结果为“2018 年 8 月 10 日 00:00:00”。

设置好时间范围后，单击按钮生成事故数量条形图，如图 10.21 所示。

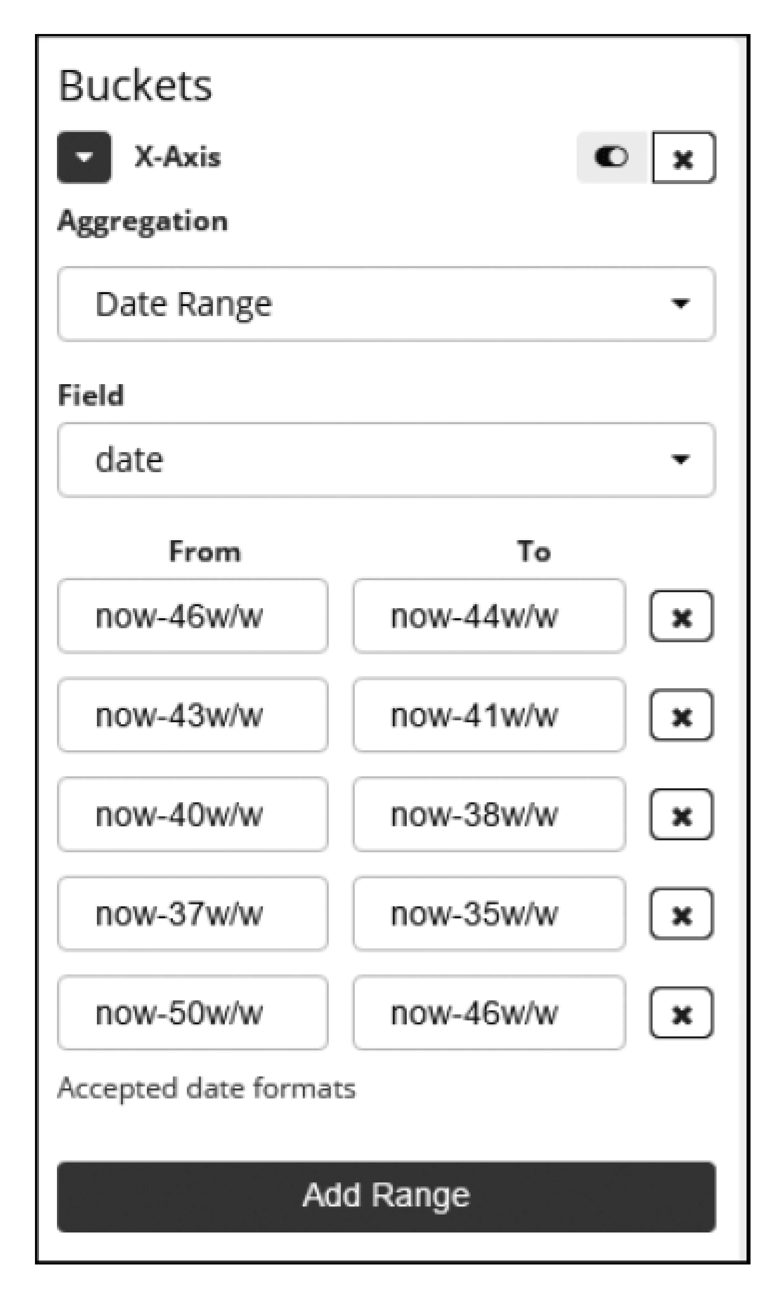


图 10.20　设置聚合方式和时间范围

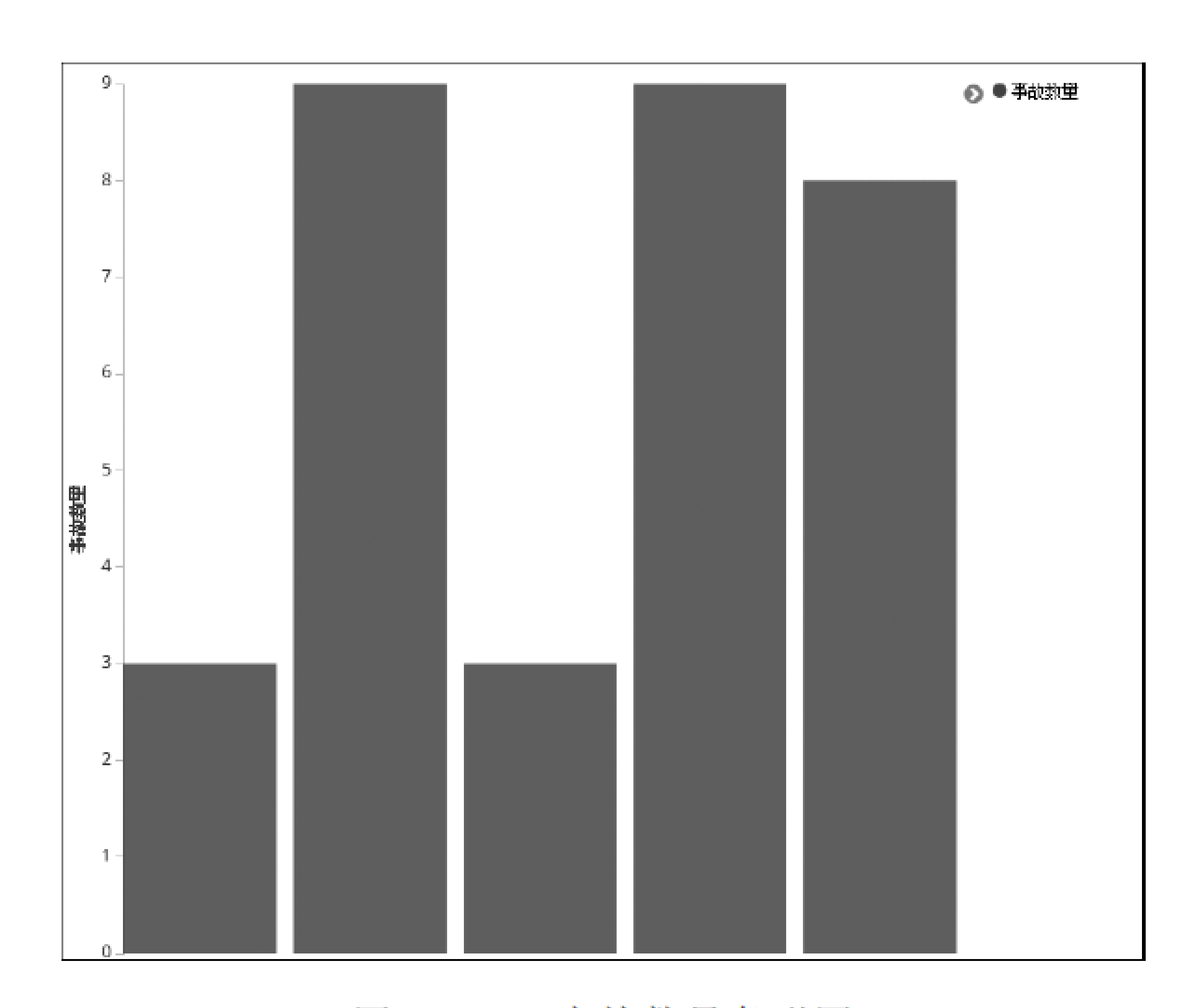


图 10.21　事故数量条形图

10.4.4　将图表集成到仪表板中

下面给出创建仪表板来放置上述图表结果的方法。在图 10.22 中，选择 Add 菜单添加

图表。

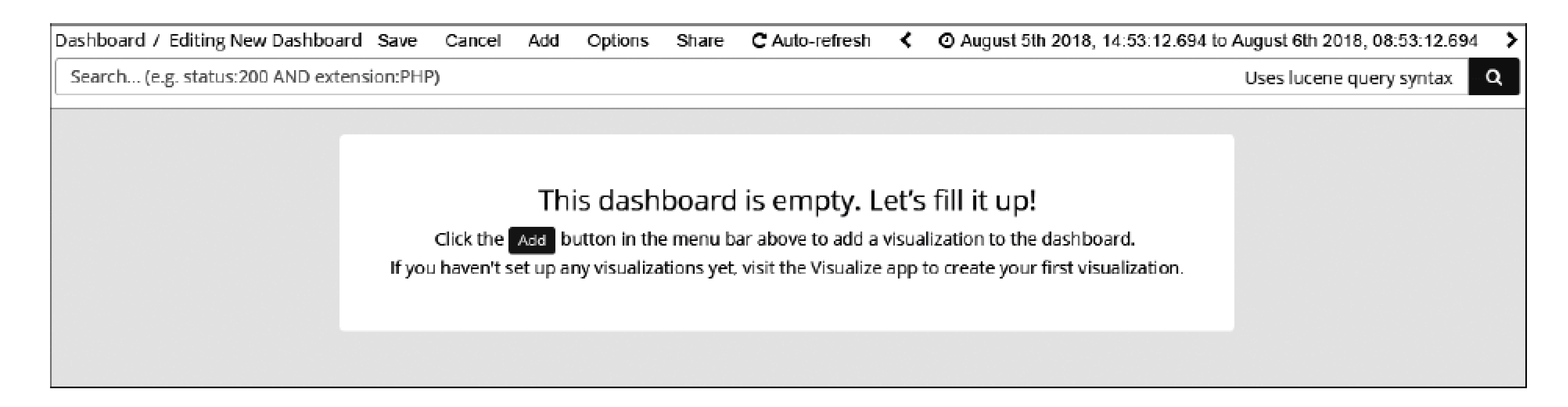


图 10.22 Dashboard 首页

将前面建立好的各个图表添加到仪表板中，如图 10.23 所示。

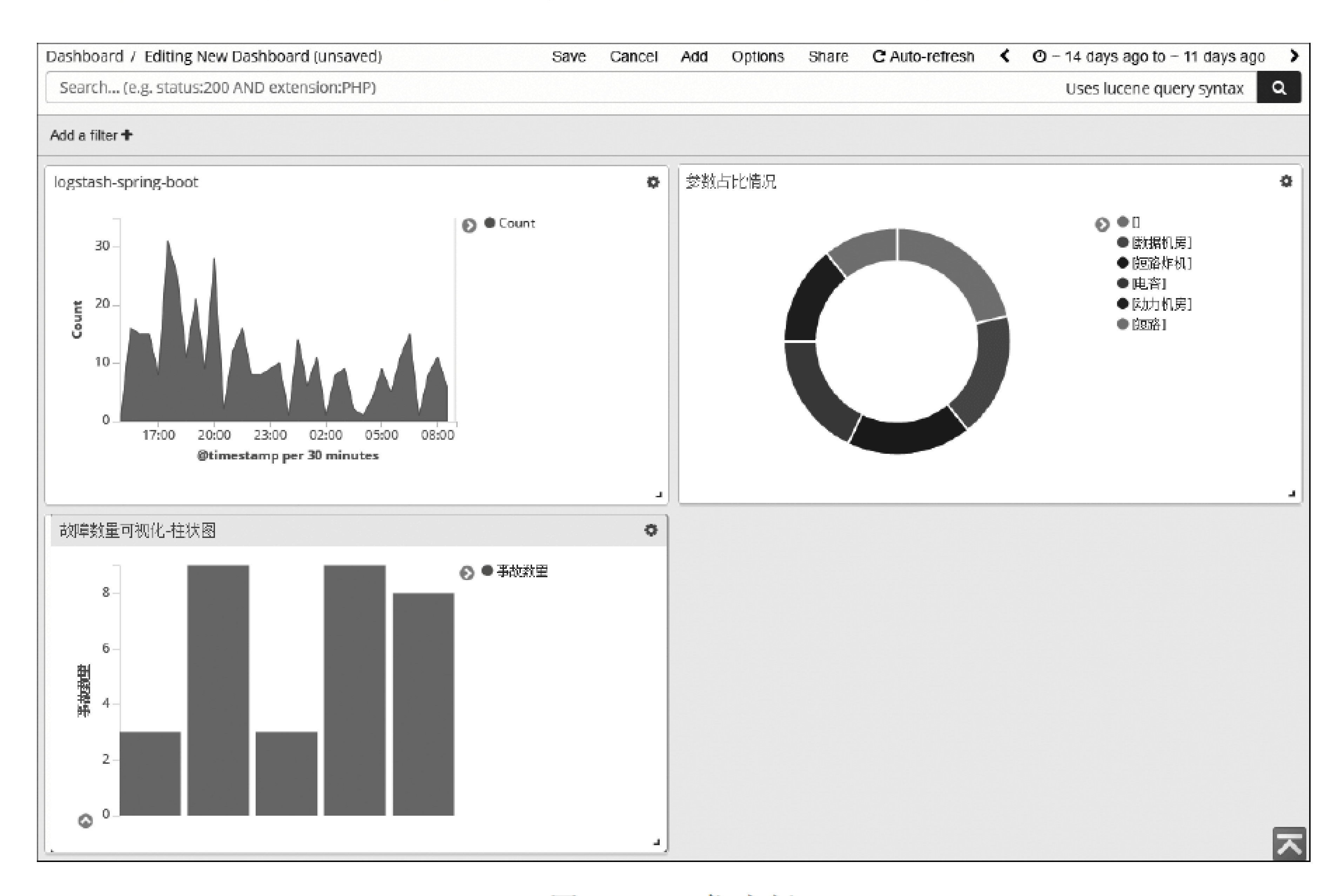


图 10.23 仪表板

10.5 扩展知识与阅读

当前 Web 服务向更精简的 DevOps 标准的流程发展。当业务逻辑复杂到一定程度时，对系统进行微服务重构，可以解耦代码和资源，让每个业务模块的构建流程更短，发布更安全，API 更稳定。要了解关于 Spring Boot 的更多信息，请参考 Spring 的官方文档(Spring, 2018)。

10.6　本章小结

本章给出了一个基于 Elasticsearch、Logstash、Kibana 的行业故障案例和规章制度的信息检索、日志分析及可视化、故障案例分析及可视化的工程实例。为使 Elasticsearch 的存储效率最大化，应对所有存储的和检索的字段进行分析，必要时对需要统计的字段开启 fielddata 功能以解析字段内容。Web 后台是基于 Spring Boot 框架实现的，前端使用 Layui 和 jQuery 完成；对日志的处理采用了 Logstash；可视化则是基于 Kibana 实现的。系统优化、故障分析可通过集成图表、可视化统计分析等方式实现。

第 11 章

Chapter 11

信息检索与分析实例(二)

> Netflix：ensuring message delivery and operational excellence；Facebook：delivering a better help experience for over a billion users；NASA JPL：powering the search for interplanetary Discovery；Shopback：smarter shopping and searching with help from the Elastic Stack.
>
> https://www.elastic.co/use-cases

在当前国内外各类信息服务和社交媒体等行业中，对于数据检索与分析的应用已经非常广泛。这一现象的背后，体现了互联网上信息服务、在线社交、新媒体等机构的爆炸式增长。同时，互联网异构数据体量巨大，随着时间的推移，有价值的数据信息日益增多，数据更新也越来越频繁，因此人们对于大数据搜索与挖掘的需求变得越来越迫切。要满足这些需求，可能会面临一些技术上的困难。首先是大数据获取。当前流行的社交网站和新媒体普遍采用各种形式的反爬策略(例如异步刷新加载数据或嵌入子页面等)，这就使得常用的开源网络爬虫框架失效。其次是各种信息汇总分析。由于对网络信息的检索并不能十分直观地描述汇总数据，因此对信息的汇总分析也是十分必要的。本章给出一个融合Elasticsearch、Logstash、Kibana、X-Pack 和 Beats 6.2 的网络信息检索与分析解决方案。信息采集部分使用网络浏览器自动化测试工具 Selenium 来完成动态网站数据的爬取[①]，项目整体基于 Spring MVC 框架开发，实现 B/S 结构的前端页面与后端 Java 程序进行数据交换、后端程序与 Elasticsearch 进行数据访问等功能。本章首先给出动态网站信息采集方法，接着完成分布式搜索，然后使用 Logstash 处理 Tomcat 服务器日志，并使用 X-Pack 和 Beats 等组件进行系统监控和数据传输，最后将所有相关数据利用 Kibana 生成统计图表和动态仪表板进行可视化展示。

---

① 本章只涉及信息采集的部分技术实现。所采集的示例数据均是网络公开数据，采集行为严格遵循相关技术规范和法规规定。

11.1 面向动态网站的信息采集

动态网站与常见的静态页面不同,它往往是将含有内容的数据通过异步刷新的方式逐步添加到前端页面中的一种网站。动态网站爬虫的编写方法与静态网站爬虫不同,一般需要针对每个不同的网站进行定制开发。这就要求编写动态网站数据爬虫的人员事先了解网站页面加载数据的机制。本节以某评测网站为例,通过模拟的方式抓取评测内容。网站中以分页的形式展示不同评测文章,爬虫程序需要自动检测页面元素的出现,完成自动翻页,打开评测内容页面,自动解析页面的评测内容,并保存爬取到的数据。

11.1.1 软件准备

在采集数据和开发搜索引擎时,需要事先安装 Elastic Stack 相关软件 Elasticsearch、Logstash 和 Kibana,并为 Elasticsearch 安装中文分析器(如 IK,详见第 2 章),为 Elasticsearch 和 Kibana安装 X-Pack 插件(详见第 8 章)。限于篇幅,对于 Beats 组件,本例中仅使用 Filebeat 和 Metricbeat。

11.1.2 浏览器驱动程序

可以利用浏览器驱动程序来模拟人对浏览器的操作,可以实现对部分网站信息的有效采集。Selenium 是一种浏览器自动化测试工具,它的运行需要两个前提条件:操作系统中装有 Selenium 支持的浏览器以及该浏览器对应的驱动程序。

以谷歌 Chrome 浏览器为例,Chrome 需要一个名为 chromedriver 的驱动程序,才能被基于 Selenium 的程序控制,并完成预先设定的操作。Chromedriver 支持多种操作系统(如 Mac OS x64、Windows x86、Linux x86 和 Linux x64),可以通过访问镜像链接 http://chromedriver.storage.googleapis.com/index.html 或 http://npm.taobao.org/mirrors/chromedriver 来获取 chromedriver 的资源。在 Linux 系统中,下载后应将 chromedriver 移动到/usr/bin 目录中;在 Windows 系统中,下载后应将 chromedriver 移动到 Chrome 浏览器目录中,并在环境变量 Path 中追加这一路径。

> Tips 上面提到的 x86 指 32 位操作系统,x64 指 64 位操作系统。另外,下载 chromedriver 程序时一定要事先查阅其版本与 Chrome 浏览器版本的对照表,只有版本相互对应的程序才可以正常使用。

11.1.3 创建索引和映像

采集数据前，需要在 Elasticsearch 中创建对应的索引和映像，并完成相应设置（分别为后面要采集的 8 种字段设置数据类型、格式和分词器等属性）。在 Kibana 的 Dev Tools 中编写代码段 11.1 所示的命令，即可创建索引和映像。

代码段 11.1：创建索引 pingce 的映像

```
PUT pingce                              #定义索引名称为 pingce
{
  "mappings": {
    "_doc": {                           #定义统一的类型名称_doc
      "properties": {
        "url": {
          "type": "keyword"
        },
        "title": {
          "type": "text",
          "index": true,
          "analyzer": "ik_max_word"
        },
        "author": {
          "type": "keyword"
        },
        "publishTime": {
          "type": "date",
          "format": "yyyy-MM-dd"        #网站中并非这一时间格式,这里由程序调整而成
        },
        "content": {
          "type": "text",
          "index": true,
          "analyzer": "ik_max_word"
        },
        "tags": {
          "type": "text",
          "index": true,
          "analyzer": "ik_max_word"
        },
        "views": {
```

```
            "type": "integer"                    #设置访问量为整数类型
          },
          "comments": {
            "type": "short"                      #设置评论数为短整型
          }
        }
      }
    }
  }
```

11.1.4　导入依赖

接下来，以手动导入依赖的方式将 Selenium、Elasticsearch、X-Pack、Gson 的依赖导入软件开发平台 IDE（如 IntelliJ IDEA）中。Selenium 全部依赖的 ZIP 包可以在其官网 https://www.seleniumhq.org/download/找到，将其中的全部 JAR 包导入即可；Elasticsearch 的大部分依赖可以在其安装目录中找到，将其中 lib 文件夹和 modules 文件夹中的所有 JAR 包导入即可（Elasticsearch 还需要导入一个 Transport 包，这个包与 Gson 包均可在阿里云的 Maven 镜像网站中央存储库中找到，地址为 http://maven.aliyun.com/mvn/view）。进入该网页后，屏幕下方的文件目录即为远程服务器中所有依赖的存放位置，如图 11.1 所示。

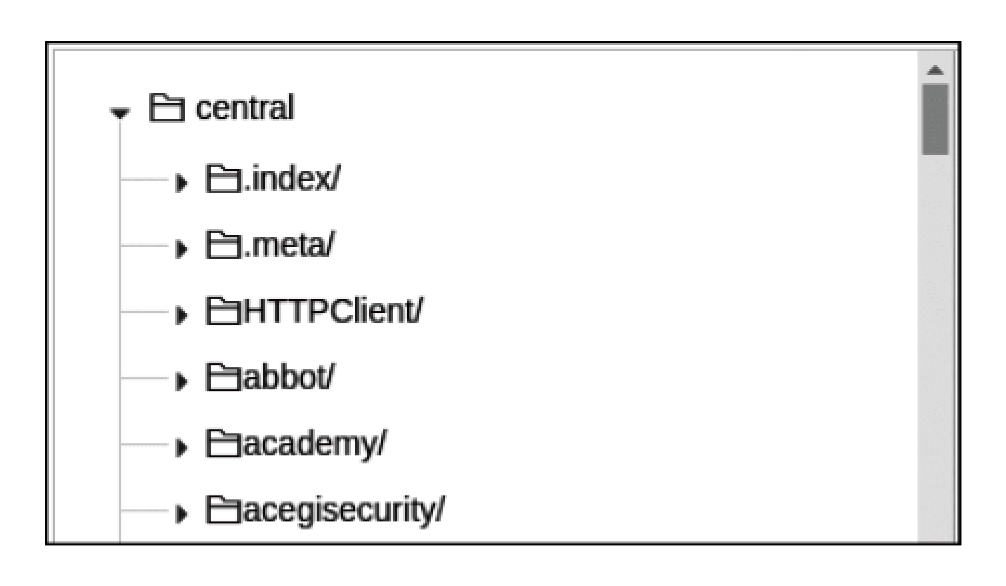


图 11.1　镜像的远程目录

展开 org/elasticsearch/client/transport/6.2.4，选择其中的 transport-6.2.4.jar，右侧会弹出该资源的基本信息，单击按钮即可获取 Elasticsearch 的 Transport 包，如图 11.2 所示。以同样的方法展开 com/google/code/gson/2.8.0，选择其中的 gson-2.8.0.jar，在弹出的详细信息中单击 Download 按钮来获取 Gson 包。

X-Pack 的依赖可以在 Elasticsearch 安装主目录下的 plugins/x-pack 目录中找到。由于 Elasticsearch 中使用了 X-Pack 插件，在创建 TransportClient 实例时还需额外引用

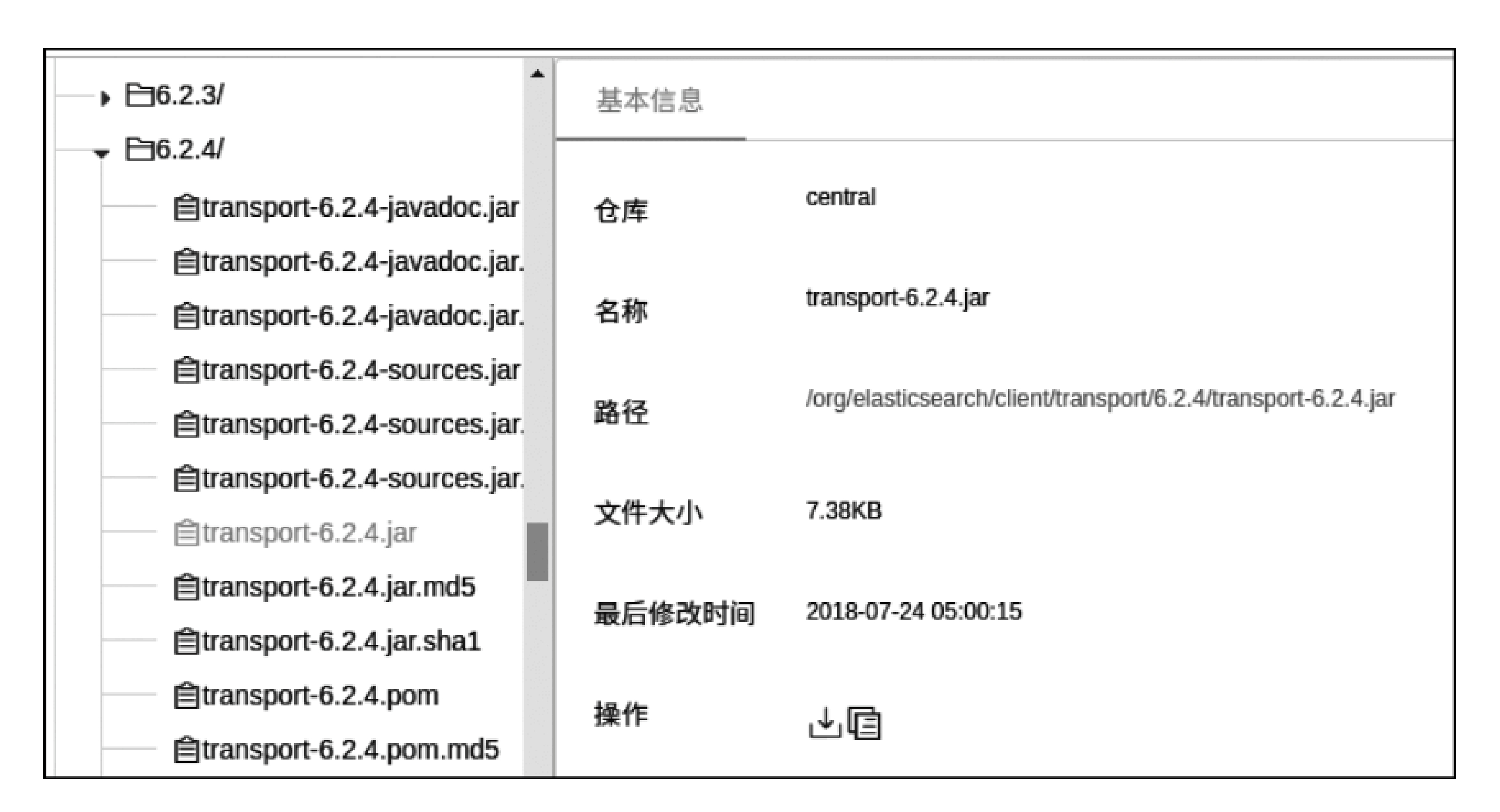


图 11.2　获取 Transport 包

x-pack-transport-6.2.4.jar 包（由于一般情况下这个包可能并未保存在 x-pack 文件夹中，因此需要访问相关的资源链接 https://artifacts.elastic.co/maven/org/elasticsearch/client/x-pack-transport/6.2.4/x-pack-transport-6.2.4.jar，以在线方式获取）。

上述工作完成后，将 3 个 JAR 包全部导入开发平台 IDEA 的 library 中即可。



资源链接中的版本号 6.2.4 也可以改为其他版本，可以根据实际情况进行修改。

11.1.5　数据采集

下面根据某网站科技产品评测页面加载数据的机制来编写相应的爬虫代码。

首先创建一个 Elasticsearch 的 Transport 实例，如代码段 11.2 所示。由于 X-Pack 需要身份验证的安全特性，Client 实例初始化时，需要在静态类 Settings 中添加 transport_admin 身份用户的验证信息（下面代码中指定的用户是 cy，配置方法详见第 8 章。如果 elasticsearch.yml 配置文件中配置了集群名称，则必须在 Settings 中添加集群名称。如果集群中开启了多个节点，那么在 addTransport Address() 方法后面应该继续链式调用 addTransportAddress() 方法将所有节点的 IP 地址和端口追加到代码中）。该初始化过程在 static 块中执行，意味着程序启动之时 Client 实例就已常驻内存，这样可以避免其反复初始化，提高运行效率。

代码段 11.2：XPackESClient 类

```
import org.elasticsearch.client.transport.TransportClient;
import org.elasticsearch.common.settings.Settings;
import org.elasticsearch.common.transport.TransportAddress;
import org.elasticsearch.transport.client.PreBuiltTransportClient;
import org.elasticsearch.xpack.client.PreBuiltXPackTransportClient;

import java.net.InetAddress;
import java.net.UnknownHostException;

public class XPackESClient {

    public static TransportClient client;

    static {
        try {
            client= new PreBuiltXPackTransportClient(Settings.builder()
                .put("cluster.name", "cyElasticsearch")
                .put("xpack.security.user", "cy:123456")
                .build())
                .addTransportAddress(new TransportAddress(InetAddress.
                getByName("localhost"), 9300));
                //如果集群中同时开启了多个节点,则应在此继续链式调用以下代码,
                //端口号递增
                //.addTransportAddress(new TransportAddress(InetAddress.
                //getByName("localhost"), 9301));
        } catch (UnknownHostException e) {
            e.printStackTrace();
        }
    }
}
```

代码段 11.3 实现了利用两个驱动程序实例分别控制两个 Chrome 浏览器的方法来对科技产品评测列表及其内容进行采集。其中 main()方法中写入了爬虫执行的主线业务逻辑，Wait4Emerging()方法实现了等待页面中某种元素加载的功能，GetReviews()方法实现了批量采集评测内容页面 URL、页面中评测标题、作者、评测正文、评测发布时间、访问量、评论数等信息的功能。

代码段 11.3：PingceCrawler 类

```
import com.google.gson.Gson;
import org.elasticsearch.action.index.IndexResponse;
import org.elasticsearch.common.xcontent.XContentType;
import org.openqa.selenium.*;
import org.openqa.selenium.chrome.ChromeDriver;
import org.openqa.selenium.interactions.Actions;
import org.openqa.selenium.support.ui.ExpectedConditions;
import org.openqa.selenium.support.ui.WebDriverWait;

import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

public class PingceCrawler {

    private static int refresh=50;          //预定翻页次数
    private static boolean isInsert=true;
    //true 表示数据直接存入 Elasticsearch;false 表示仅测试,不入库
    private static WebDriver driver, innerDriver;
    private static WebDriverWait wait;
    private static List<String> urlList=new ArrayList<>();
    private static String url;
    private static String title;
    private static String author;
    private static String publishTime;
    private static List<WebElement> elementList;
    private static StringBuilder builder=new StringBuilder();
    private static String content;
    private static String tags;
    private static String views;
    private static String comments;
    private static Actions actions;

    public static void main(String[] args) {
        //初始化谷歌 Chrome 浏览器驱动程序
        driver=new ChromeDriver();
        //访问评测页面
```

```
        driver.get("url");    //这里写入待采集的网站的 URL 地址
        //等待评测信息加载,完成的标志是"下一页"按钮出现
        Wait4Emerging("//a[@ class= 'next']");
        //获取当前页 12 条评测的链接
        for (int i= 1; i<= 12; i++) {
            urlList.add(driver.findElement(By.xpath("//ul[@ id= 'post_container']/li["+ i+ "]
            /div[@ class= 'article']/h2/a")).getAttribute("href"));
        }
        //获取该 12 条评测的数据
        GetReviews(urlList);
        urlList.clear();
        for (int i= 0; i< refresh; i++) {
            //单击"下一页"按钮翻页
            driver.findElement(By.xpath("//a[@ class= 'next']")).click();
            for (i= 1; i<= 12; i++) {
                try {
                    urlList.add(driver.findElement(By.xpath("//ul[@ id= 'post_container']/li["+ i
                    + "]
                    /div[@ class= 'article']/h2/a")).getAttribute("href"));
                } catch (NoSuchElementException nsee) {
                    //获取不到后续评测,表示已经结束,退出
                    System.exit(0);
                }
            }
            GetReviews(urlList);
            urlList.clear();
        }
    }

    public static void Wait4Emerging(String strXpath) {
        while (true) {
            try {
                wait= new WebDriverWait(driver, 5);
                wait.until(ExpectedConditions.presenceOfElementLocated(By.
                xpath(strXpath)));
                if (driver.findElement(By.xpath(strXpath)) != null)
                    break;
            } catch (NoSuchElementException nsee) {
                System.out.println("检测的信息未找到,继续等待。");
```

```
            } catch (TimeoutException te) {
                System.out.println("检测过程超出 5s 预定时间,继续等待。");
            }
        }
    }

    public static void GetReviews(List<String> urlList) {
        //初始化抓取评测内容的谷歌 Chrome 浏览器驱动程序
        innerDriver=new ChromeDriver();
        for (String link : urlList) {
            innerDriver.get(link);
            url=link;
            System.out.println(link);
            try {
                title=innerDriver.findElement(By.xpath("//div[@class=
                'article_container row  box']/h1")).getText();
                System.out.println("title: "+title);
                author=innerDriver.findElement(By.xpath("//span[@class='info_author info_ico
                ']")).getText();
                System.out.println("author: "+author);
                publishTime=innerDriver.findElement(By.xpath("//span[@class='info_date info_ico
                ']")).getText();
                //将时间格式调整为年-月-日的格式,此处可根据实际情况定制
                if (Integer.parseInt(publishTime.substring(0, 2))>=10)
                publishTime="2017-"+publishTime;
                else publishTime="2018-"+publishTime;
                System.out.println("publishTime: "+publishTime);
                elementList=innerDriver.findElements(By.xpath("//div[@id=
                'post_content']/p"));
                //去掉没有价值的部分信息
                for (WebElement element : elementList) {
                    if (element.getText().trim().equals("")) continue;
                    if (element.getText().trim().equals("评测视频：")) continue;
                    if (element.getText().trim().contains("<span>") &&
                    element.getText().trim().contains("</span>"))
                        continue;
                    builder.append(element.getText()+"\n");
                }
                content=builder.toString();
```

```
                System.out.println("content:"+ content);
                builder.delete(0, builder.length()- 1);
                 List< WebElement> tagsList= innerDriver.findElements(By.xpath("//div[@ class= '
                 tagcloud']/a"));
                for (WebElement tag : tagsList) builder.append(tag.getText()+ " ");
                builder.deleteCharAt(builder.length()- 1);
                tags= builder.toString();
                System.out.println("tags: "+ tags);
                if (builder.length()> 0)
                    builder.delete(0, builder.length()- 1);
                 views= innerDriver.findElement(By.xpath("//span[@ class= 'info_views info_ico
                 ']")).getText();
                System.out.println("views: "+ views);
                comments= innerDriver.findElement(By.xpath("//span[@ class=
                'info_comment info_ico']/a")).getText();
                System.out.println("comments: "+ comments);
                if (isInsert) {
                    Map< String, Object> map= new HashMap< > ();
                    map.put("url", url);
                    map.put("title", title);
                    map.put("author", author);
                    map.put("publishTime", publishTime);
                    map.put("content", content);
                    map.put("tags", tags);
                    map.put("views", views);
                    map.put("comments", comments);
                    String s= new Gson().toJson(map);
                    IndexResponse response= ESClient.client.prepareIndex
                    ("pingce", "_doc")
                            .setSource(s, XContentType.JSON).get();
                }
            } catch (NoSuchElementException nse) {
                //防止页面出错导致程序中断
                continue;
            }
        }
        //按组合键 Alt+ F4 关闭当前窗口
        actions= new Actions(innerDriver);
        actions.keyDown(Keys.ALT).sendKeys(Keys.F4).keyUp(Keys.ALT).perform();
```

```
            innerDriver.close();
            innerDriver.quit();
        }
    }
```

要运行 Selenium 采集程序，需要先启动浏览器驱动程序。进入/usr/bin 目录，在终端执行命令./chromedriver 启动 chromedriver，如图 11.3 所示。

```
cy@cy-N53SN:~$ cd /usr/bin
cy@cy-N53SN:/usr/bin$ ./chromedriver
Starting ChromeDriver 2.41.578700 (2f1ed5f9343c13f73144538f15c00b370eda6706) on
port 9515
Only local connections are allowed.
```

图 11.3　启动 Chromedriver

最后，编译并运行 Java 程序，如果程序执行正常，浏览器会自动弹出，并自动执行操作，采集到的信息会在存入 Elasticsearch 的同时输出到 IDEA 的控制台。采集完成后，可以在 Kibana 中添加索引 pingce 的索引模式，在 Discover 程序中可以查看 Elasticsearch 索引中的数据，图 11.4 为采集的部分数据。

Time	_source
September 1st 2018, 08:00:00.000	publishTime: September 1st 2018, 08:00:00.000 comments: 5 author: 大米评测 title: 小米5X体验评测（对比魅蓝X、小米6） url: [illegible] content: 6Hello，大家好我是大米，又和大家见面啦，最近有不少小伙伴问，诶，大米你能不能做一个有关小米5X的评测视频？因为新版本4+32GB售价1299看起来好像还可以，于是大米也是应小伙伴的要求，买了一台，那么这台机子实际表现到底怎么样？今天我们就一起来简单了解下吧~ 视频评测： 产品外观： 评测小结： 简单使用几天之后的一点小感受就是，小米5X在做工质

图 11.4　Elasticsearch 中存储的部分数据

11.2　基于 Spring MVC 的信息检索及 Web 程序设计

11.2.1　创建和配置 Spring MVC 项目

在 IDEA 中创建一个 Spring MVC 项目（本例中设置项目名称为 PingceSearch）。在 IDEA 的 library 中添加 Spring Framework 的全部依赖，如果 IDEA 并没有自动下载这些依赖，可以访问 Spring 官网来获取，相关页面的 URL 为 http://projects.spring.io/spring-framework/。打开 web.xml 配置文件，在其中的<web-app></web-app>标记中间添加如代码段 11.4 所示的配置。其中标记<context-param></context-param>中指定了配置文件 dispatcher-servlet.xml 的存放位置，<listener></listener>中指定了自动装配

dispatcher-servlet.xml 中配置信息的监听器，<servlet></servlet>中指定了处理映射的分发器，<servlet-mapping></servlet-mapping>指定了要匹配的路径。

代码段 11.4：Web.xml 配置文件中添加的配置

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns="http://xmlns.jcp.org/xml/ns/javaee"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
         http://xmlns.jcp.org/xml/ns/javaee/web-app_3_1.xsd"
         version="3.1">
    <context-param>
        <param-name>contextConfigLocation</param-name>
        <param-value>/WEB-INF/dispatcher-servlet.xml</param-value>
    </context-param>
    <listener>
        <listener-class>org.springframework.web.context.ContextLoaderListener
        </listener-class>
    </listener>
    <servlet>
        <servlet-name>dispatcher</servlet-name>
        <servlet-class>org.springframework.web.servlet.DispatcherServlet
        </servlet-class>
        <load-on-startup>1</load-on-startup>
    </servlet>
    <servlet-mapping>
        <servlet-name>dispatcher</servlet-name>
        <url-pattern>/</url-pattern>
    </servlet-mapping>
</web-app>
```

在存放 Java 代码的文件夹（如 src 文件夹）创建一个自定义反转域名的包（本例中定义为 com.cy.controller，用于存放属于自己的控制器类）。在 WEB-INF 文件夹中创建一个 views 文件夹用于存放前端界面。在 WEB-INF 文件夹中添加一个 dispatcher-servlet.xml 配置文件用于配置自定义反转域名包的扫描路径、启用注解、前端页面（即视图）的前缀和后缀等。配置文件的内容如代码段 11.5 所示。

代码段 11.5：dispatcher-servlet.xml 配置文件

```
<?xml version="1.0" encoding="UTF-8"?>
```

```
<beans xmlns="http://www.springframework.org/schema/beans"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xmlns:context="http://www.springframework.org/schema/context"
        xmlns:mvc="http://www.springframework.org/schema/mvc"
        xsi:schemaLocation="http://www.springframework.org/schema/beans
        http://www.springframework.org/schema/beans/spring-beans.xsd
        http://www.springframework.org/schema/context
        http://www.springframework.org/schema/context/spring-context.xsd
        http://www.springframework.org/schema/mvc
        http://www.springframework.org/schema/mvc/spring-mvc.xsd">

    <context:component-scan base-package="com.cy.controller"></context:
    component-scan>

    <mvc:annotation-driven/>
    <mvc:resources mapping="/js/**" location="/js/"/>

    <bean id="ViewResolver" class="org.springframework.web.servlet.view.
    InternalResourceViewResolver">
        <property name="prefix" value="/WEB-INF/views"/>
        <property name="suffix" value=".jsp"/>
    </bean>
</beans>
```

上述配置完成后，整个项目的文件目录结构如图 11.5 所示。

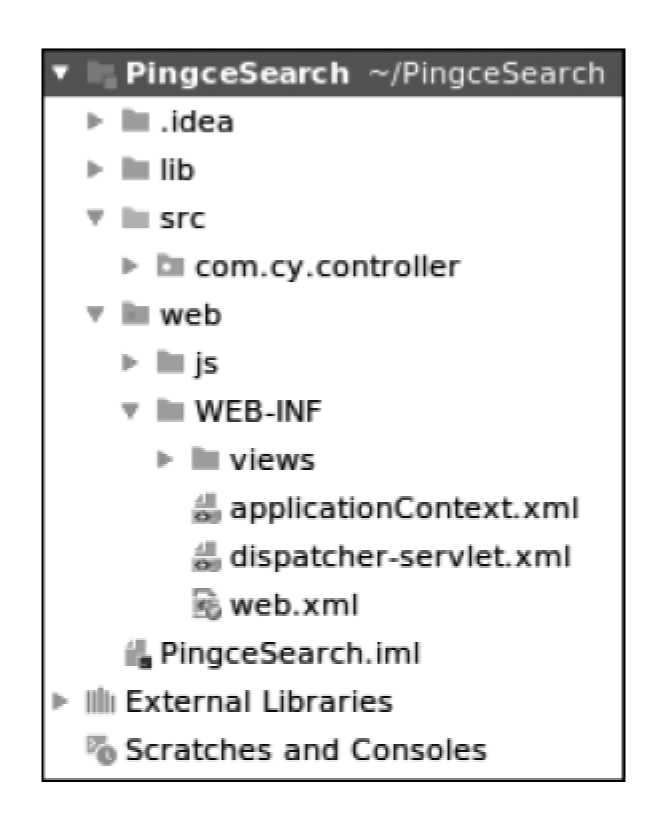


图 11.5　PingceSearch 项目的文件目录结构

11.2.2 前端页面设计

在程序前端初始页面中放置一个搜索词输入框和一个“搜索”按钮，如图 11.6 所示。

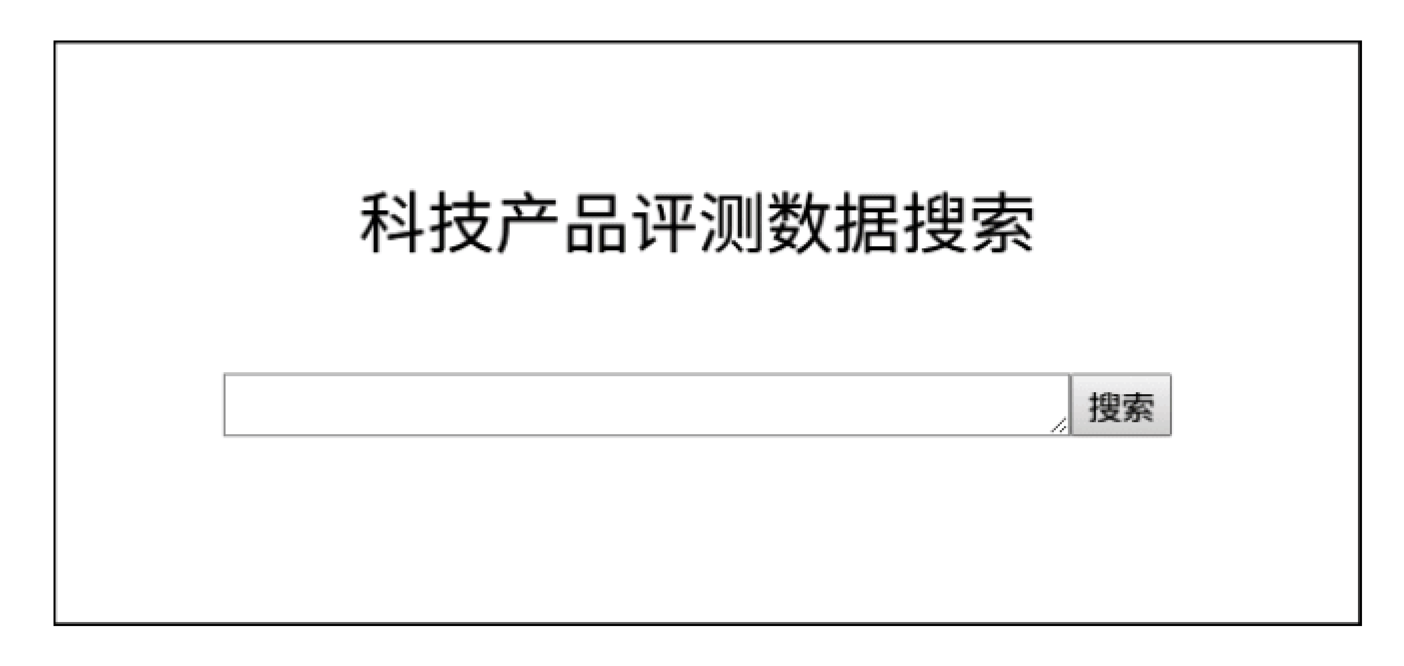


图 11.6 前端初始页面

用户在输入框中输入搜索词后，按回车键或单击“搜索”按钮，前端 jQuery 采集用户输入的搜索词，通过 Ajax 程序传到后端 Java 控制器程序。控制器与 Elasticsearch 交互，将查询结果处理后返回到前端。最后，前端通过异步刷新将数据输出到界面中。在 WEB-INF/views 文件夹中创建前端页面 index.jsp，该 JSP 文件内容如代码段 11.6 所示。

代码段 11.6：前端页面 index.jsp

```
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<html>
<head>
    <title>科技产品评测数据搜索</title>
    <style type="text/css">
        body {
            margin: 0px auto;
            padding: 0px;
        }

        #main, #result, #detail {
            width: 800px;
            margin: 0px auto;
            padding: 0px;
            text-align: left;
            line-height: 35px;
        }

        #result, #detail {
```

```
    display: none;
    margin-top: 50px;
}

#title {
    width: 800px;
    margin: 150px auto 30px auto;
    padding: 0px;
    font-size: 24pt;
    text-align: center;
    line-height: 80px;
}

#keyword {
    width: 800px;
    margin: 0px auto;
    padding: 0px;
    font-size: 12pt;
    text-align: center;
    vertical-align: middle;
    line-height: 25px;
}

#kw {
    width: 400px;
    height: 30px;
    line-height: 25px;
    padding-left: 5px;
    white-space: nowrap;
    overflow: hidden;
    vertical-align: middle;
}

#submit {
    tab-index: 1;
    font-size: 12pt;
    vertical-align: middle;
}

.a_title {
```

```
        text-decoration: underline;
        color: blue;
        font-size: 12pt;
        line-height: 20px;
    }

    .publishTime {
        color: #555;
        line-height: 25px;
        font-size: 9.75pt;
    }

    .content {
        color: #222;
    }

    .a_url {
        text-decoration: none;
        color: green;
        font-size: 9.75pt;
        line-height: 20px;
    }

    .d_title {
        font-size: 20pt;
        text-align: center;
        line-height: 50px;
    }

    .d_content {
        font-size: 9.75pt;
        line-height: 20px;
    }

    spot {
        color: red;
    }

    p {
        margin: 0px;
```

```
            padding: 0px;
            font-size: 9.75pt;
            line-height: 20px;
        }
    </style>
    <script src="/pingcesearch/js/jquery-1.11.3.min.js" type="text/
    javascript"></script>
    <script type="text/javascript">
        $(document).ready(function () {
            $("#kw").focus();
            $("#submit").click(function () {
                var query=$('#kw').val();
                $.ajax({
                    type: 'POST',
                    contentType: 'text/html;charset=UTF-8',
                    url: '/pingcesearch/search',
                    data: query,
                    dataType: 'text',
                    success: function (data) {
                        $('#list').empty();
                        $('#list').append(decodeURI(data));
                        $('#main').slideToggle();
                        $('#result').slideToggle();
                    },
                    error: function (textStatus) {
                        alert("Error:"+textStatus);
                    }
                });
            });
            //返回首页按钮事件
            $("#back2index").click(function () {
                $("#main").slideToggle();
                $("#result").slideToggle();
            });
            //返回列表按钮事件
            $("#back2list").click(function () {
                $("#result").slideToggle();
                $("#detail").slideToggle();
            });
        });
```

```
        //按回车键直接执行搜索
        function exec_search() {
            if (window.event.keyCode==13) {
                if (document.all('submit') != null) {
                    document.all('submit').click();
                }
            }
        }

        function viewDetail(strId) {
            $.ajax({
                type: 'POST',
                contentType: 'text/html;charset=UTF-8',
                url: '/pingcesearch/detail/',
                data: strId,
                dataType: 'text',
                success: function (data) {
                    $('#item').empty();
                    $('#item').append(decodeURI(data));
                    $('#result').slideToggle();
                    $('#detail').slideToggle();
                    $(window).scrollTop(0);
                },
                error: function (textStatus) {
                    alert("Error:"+ textStatus);
                }
            });
        }
    </script>
</head>
<body onkeydown="exec_search();">
<div id="main">
    <div id="title">
        科技产品评测数据搜索
    </div>
    <div id="keyword">
        <input id="kw" type="text"><input type="button" id="submit" value="搜索">
    </div>
</div>
```

```
<div id="result">
    <input type="button" id="back2index" value="返回首页"><br/><br/>
    <div id="list">
    </div>
</div>
<div id="detail">
    <input type="button" id="back2list" value="返回列表"><br/><br/>
    <div id="item">
    </div>
</div>
</body>
</html>
```

Tips 前端页面引用的 jquery-1.11.3.min.js 文件是一个普通的 jQuery 库，以静态文件的形式存放在 web/js 目录中，其静态文件访问配置已写在 dispatcher-servlet.xml 配置文件中。

11.2.3 后端控制器类

默认情况下，程序不能直接访问 index 页面，需要在 com.cy.controller 包中编写一个 HomeController 类，以便项目运行时可以直接跳转到 index 页面，该类的代码如代码段 11.7 所示。

代码段 11.7：跳转到 index 页面的 HomeController 类

```
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;

@Controller
public class HomeController {
    @RequestMapping(value="/")
    public String home() {
        return "/index";
    }
}
```

在 index 页面中，Ajax 程序指定了数据交换的控制器路径为/search，则该页面的后端控制器类应设置请求映射为/search 并与 index 页面之间进行数据收发。在 com.cy.

controller 包中创建该页面对应的控制器类，该类的实现如代码段 11.8 所示，其中包含了两个提供搜索功能的方法（第一个负责查询搜索结果列表的数据，第二个负责查询一条结果中所有详细内容的数据）。

代码段 11.8：SearchController 类

```
import org.elasticsearch.action.get.GetResponse;
import org.elasticsearch.action.search.SearchResponse;
import org.elasticsearch.action.search.SearchType;
import org.elasticsearch.index.query.QueryBuilder;
import org.elasticsearch.search.SearchHit;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.* ;

import java.io.UnsupportedEncodingException;
import java.net.URLEncoder;
import java.util.Map;

import static com.cy.controller.XPackESClient.client;
import static org.elasticsearch.index.query.QueryBuilders.queryStringQuery;

@Controller
public class SearchController {

    @RequestMapping(value= "/search", method= RequestMethod.POST)
    @ResponseBody
    public String search(@RequestBody String query) {
        QueryBuilder qb= queryStringQuery("title:"+ query);
        SearchResponse response= client.prepareSearch("pingce")
                .setTypes("_doc")
                .setSearchType(SearchType.DFS_QUERY_THEN_FETCH)
                .setQuery(qb)
                .setFrom(0)
                .setSize(10)
                .setExplain(true)
                .get();
        SearchHit[] hits= response.getHits().getHits();
        Map<String, Object> currentHit;
        String _Id, title, publishTime, content, url;
        StringBuilder builder= new StringBuilder();
```

```
        for (SearchHit hit : hits) {
            _Id= hit.getId();
            currentHit= hit.getSourceAsMap();
            publishTime= currentHit.get("publishTime").toString();
            title= currentHit.get("title").toString().replaceAll(query,
            "<spot>"+ query+ "</spot>");
            url= currentHit.get("url").toString();
            content= currentHit.get("content").toString().replaceAll(query,
            "<spot>"+ query+ "</spot>");
            builder.append("<a class= \"a_title\" href= \"javascript:void(0)\" onclick= \"
            viewDetail('"+ _Id+ "')\" target= \"view_window\">"+ Utf8(title)+ "</a>");
            builder.append("<p class= \"publishTime\">"+ Utf8("发布日期：")+
            publishTime+ "</p>");
            builder.append("<p>"+ Utf8(content.length()> 200 ? content.
            substring(0, 200)+ "…" : content)+ "</p>");
            builder.append("<a class= \"a_url\" href= \""+ url+ "\" target= \"view_
            window\">"+ url.substring(url.indexOf(":")+ 3)+ "</a><br/><br/>");
        }
        return builder.toString();
    }

    @RequestMapping(value= "/detail", method= RequestMethod.POST)
    @ResponseBody
    public String detail(@RequestBody String _Id) {
        GetResponse response= client.prepareGet("pingce", "_doc", _Id).get();
        Map<String, Object> hit= response.getSourceAsMap();
        String author, views, comments, publishTime, title, url, content, tags;
        StringBuilder builder= new StringBuilder();
        author= hit.get("author").toString();
        publishTime= hit.get("publishTime").toString();
        views= hit.get("views").toString();
        comments= hit.get("comments").toString();
        title= hit.get("title").toString();
        url= hit.get("url").toString();
        content= hit.get("content").toString();
        tags= hit.get("tags").toString();
        builder.append("<p class= \"d_title\">"+ Utf8(title)+ "</p>");
        builder.append("<p class= \"d_content\">"+ Utf8("编辑："+ author)+ "</p>");
```

```
        builder.append("<p class=\"d_content\">"+Utf8("发表时间：")+
        publishTime+"</p>");
        builder.append("<p class=\"d_content\">"+Utf8("访问量：")+views+
        "        ");
        builder.append(Utf8("评论数：")+comments+"</p>");
        builder.append("<p class=\"d_content\">"+Utf8("来源：")+"<a href=\"
        "+url+"\" target=\"view_window\">"+url+"</a></p>");
        builder.append("<p class=\"d_content\">"+Utf8(content)+"</p>");
        builder.append("<p class=\"d_content\">"+Utf8("标签："+tags)+
        "</p>");
        return builder.toString();
    }

    public String Utf8(String input) {
        String output="";
        try {
            output=URLEncoder.encode(input, "utf-8");
        } catch (UnsupportedEncodingException e) {
            e.printStackTrace();
        }
        return output.replaceAll("\\+", " ").replaceAll("%20", " ").replaceAll("%21", "!")
                .replaceAll("%22", "\"").replaceAll("%23", "#").replaceAll
                ("%24", "$")
                .replaceAll("%25", "%").replaceAll("%26", "&").replaceAll
                ("%27", "\'")
                .replaceAll("%28", "(").replaceAll("%29", ")").replaceAll
                ("%2A", "*")
                .replaceAll("%2B", "+").replaceAll("%2C", ",").replaceAll
                ("%2D", "-")
                .replaceAll("%2E", ".").replaceAll("%2F", "/").replaceAll
                ("%3A", ":")
                .replaceAll("%3B", ";").replaceAll("%3C", "<").replaceAl
                l("%3D", "=")
                .replaceAll("%3E", ">").replaceAll("%3F", "?").replaceAll
                ("%40", "@")
                .replaceAll("%5B", "[").replaceAll("%5C", "\\").replaceAll
                ("%5D", "]")
                .replaceAll("%5E", "^").replaceAll("%5F", "_").replaceAll
                ("%60", "`")
```

```
                        .replaceAll("%7B", "{").replaceAll("%7C", "|").replaceAll
                        ("%7D", "}")
                        .replaceAll("%7E", "~");
        }
    }
```

对上述程序进行编译，打成 WAR 包放到 Tomcat 服务器程序主目录下的 webapps 文件夹中。接着启动 Tomcat 服务器，在浏览器地址栏中输入 http://localhost:8080/pingcesearch，即可看到项目的首页。例如，输入搜索词“魅族”并单击“搜索”按钮，页面中即出现对应的搜索结果，同时搜索词在搜索结果的标题和正文摘要中被高亮，如图 11.7 所示。由于该搜索程序全程使用 Ajax 技术执行，页面显示搜索结果的过程是在单个页面中执行的，并没有发生跳转。

返回首页

拍照旗舰？魅族MX6体验评测

发布日期：2018-08-02

OHello，大家好我是大米，又和大家见面啦，那么今天大米要给大家带来的是魅族MX6这款手机的体验评测，说实话对于这款手机来说，大米可能比较关心它的相机表现到底如何，那么接下来我们就一起来看下吧！ 视频评测： 产品外观： 今年的MX6在外包装上相比以往的MX系列进行还是有明显变化的，从原来的书册形式变成了磁吸式的包裹，包装正面印了一个手机的轮廓；整体的外观设计和之前的P...

图 11.7　搜索结果列表页面

在列表中单击任意一条搜索结果的标题，页面中会显示出该评测的详细信息。

11.3　基于 Logstash 的日志处理

本节通过配置相关文件来控制 Logstash 收集 Tomcat 服务器用户访问日志，并将其作为输入。使用 grok 过滤器来解析日志内容，并将处理好的数据输出到终端和 Elasticsearch。在 Logstash 安装主目录下的执行目录 bin 中创建一个配置文件 conf.conf，以便供 Logstash 运行时加载。配置文件的内容如代码段 11.9 所示。由于当前的 Elasticsearch 集群配置了 X-Pack，需要身份验证才能访问其中的数据，因此在配置 Logstash 输出到 Elasticsearch 时，需要额外配置用于身份验证的账号和密码。

代码段 11.9：控制 Logstash 输入、过滤、输出的配置文件 conf.conf

```
input {                        #配置输入
    file {                     #从文件输入
```

```
            path=> ["/usr/apache-tomcat-7.0.90/logs/localhost_access_log.txt"]
        }
    }
    filter {                        #配置过滤
        grok {
            match=> {
                "message"=> "%{IPORHOST:clientip} %{USER:ident} %{USER:auth}
                \[%{HTTPDATE:timestamp}\]
                \"(?:%{WORD:verb} %{URIPATHPARAM:request} (?: HTTP/%{NUMBER:httpversion})?|-)\" %
               {NUMBER:response}
                %{NUMBER:responsetime}?|-"
            }
        }
    }
    output {                        #配置输出
        stdout {
            codec=> rubydebug
        }
        elasticsearch {
            hosts=> ["localhost"]
            user=> "elastic"        #指定 Elasticsearch 安全集群的用户身份验证信息
            password=> "elastic"
        }
    }
```

保存配置文件 conf.conf,使用终端命令./logstash -f ./conf.conf 启动 Logstash,同时加载配置文件。自 Logstash 启动之时起,每当 Tomcat 服务器日志增加新内容,就会被 Logstash 获取、处理并传入 Elasticsearch。

11.4 基于 Beats 的数据传输

本节使用 Filebeat 和 Metricbeat 来传输日志文件和系统指标数据。首先,在 Filebeat 和 Metricbeat 的配置文件中修改配置信息,使其拥有向 Elasticsearch 传输数据的权限。打开 Filebeat 的配置文件/etc/filebeat/filebeat.yml,将 output.elasticsearch 一节中注释掉的 username 和 password 两行取消注释,并根据实际情况修改这部分最后两行的身份验证信息。对 Metricbeat 的配置文件/etc/metricbeat/metricbeat.yml 也应进行类似的修改。这两个配置文件中修改后的 Elasticsearch 输出部分如代码段 11.10 所示。

代码段 11.10：配置文件中 Elasticsearch 输出部分

```
#------------------Elasticsearch output ----------------------
output.elasticsearch:
  #Array of hosts to connect to.
  hosts: ["localhost:9200"]

  #Optional protocol and basic auth credentials.
  #protocol: "https"
  username: "elastic"
  password: "elastic"
```

修改配置文件后即可运行 Filebeat 和 Metricbeat。执行终端命令 sudo /etc/init. d/filebeat start 来启动 Filebeat，执行终端命令 sudo /etc/init. d/metricbeat start 来启动 Metricbeat。

11.5 基于 Kibana 的数据可视化

启动 Kibana，以超级管理员身份登录，在 Management 组件的 Create index pattern 界面中输入索引名称 pingce 来创建索引模式，如图 11. 8 所示。要创建的索引模式还包括 logstash-＊、filebeat-＊ 和 metricbeat-＊，均以此方法添加。

图 11. 8　创建索引模式

11.5.1　可视化索引文件中的信息

下面对索引 pingce 中的文档数据进行统计。首先创建一个面积图，在 Visualize 中选择 Area chart 选项，设置横轴聚合方式为 Date Histogram，在 Metrics & Axes 的 Line Mode 中选择 stepped，其他选项保留默认设置，单击按钮生成统计图，如图 11.9 所示。该图直观地反映了评测文章发表的数量随时间的变化情况。

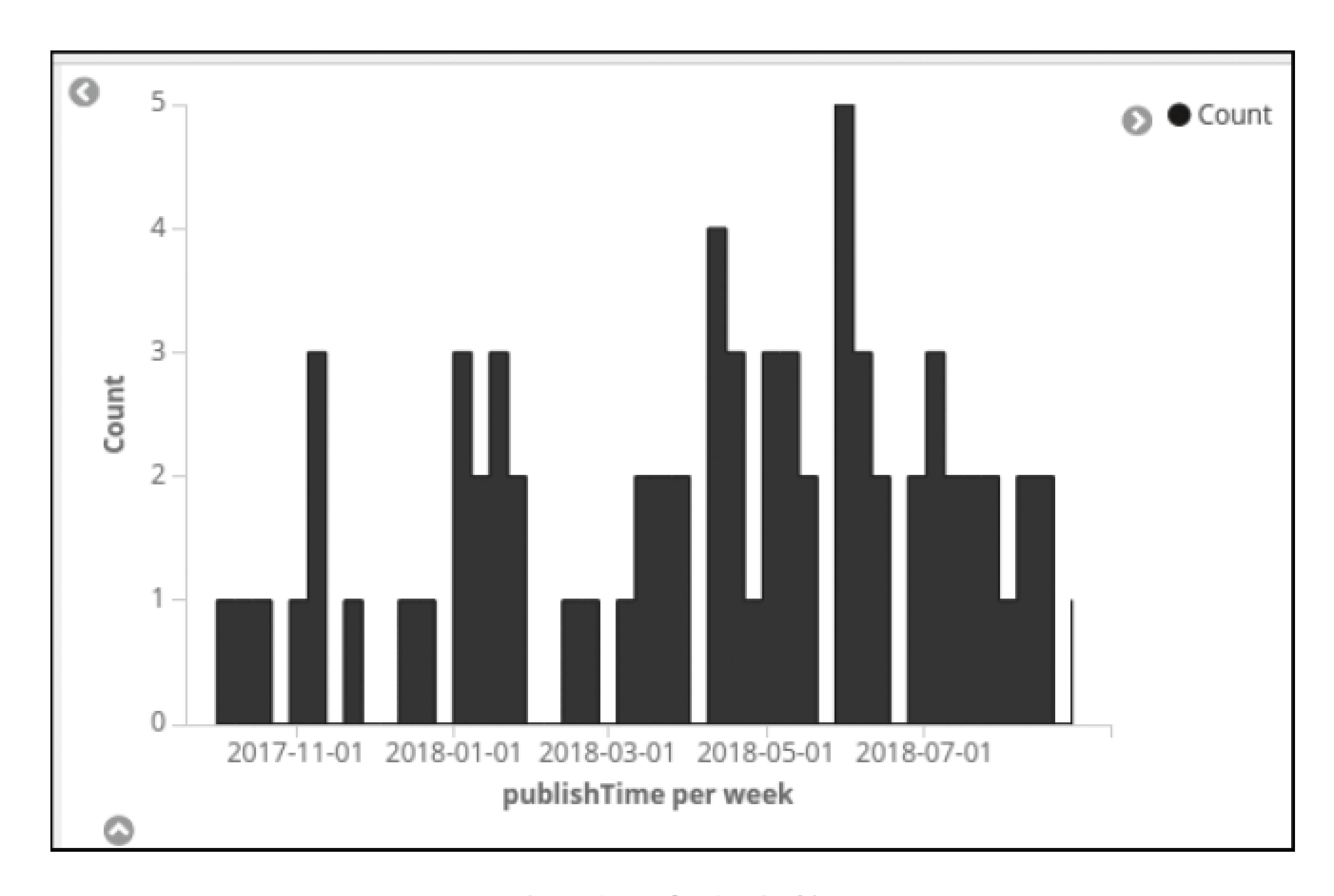


图 11.9　评测文章发表数量面积图

接着对该索引中的两种数值型数据访问量和评论数按不同的计数范围进行统计。这里以访问量为例创建一个饼图，在 Buckets 设置中选择聚合方式 Range，在 Field 中选择字段 views，并在下方设置多个计数范围，如图 11.10 所示。

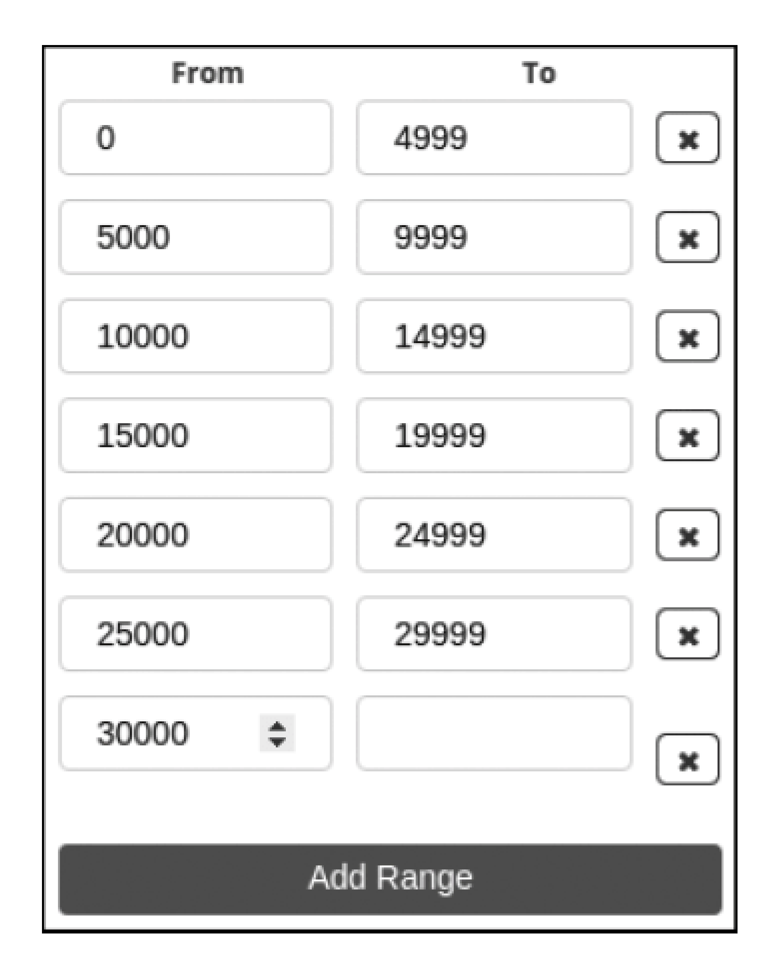


图 11.10　为饼图设置聚合的计数范围

完成上述设置后，即可得到如图 11.11 所示的饼图。从图中可以看出，评测文章的访问量大多在 15 000 次以内，其中获得 5000～10 000 次访问的评测文章最多。

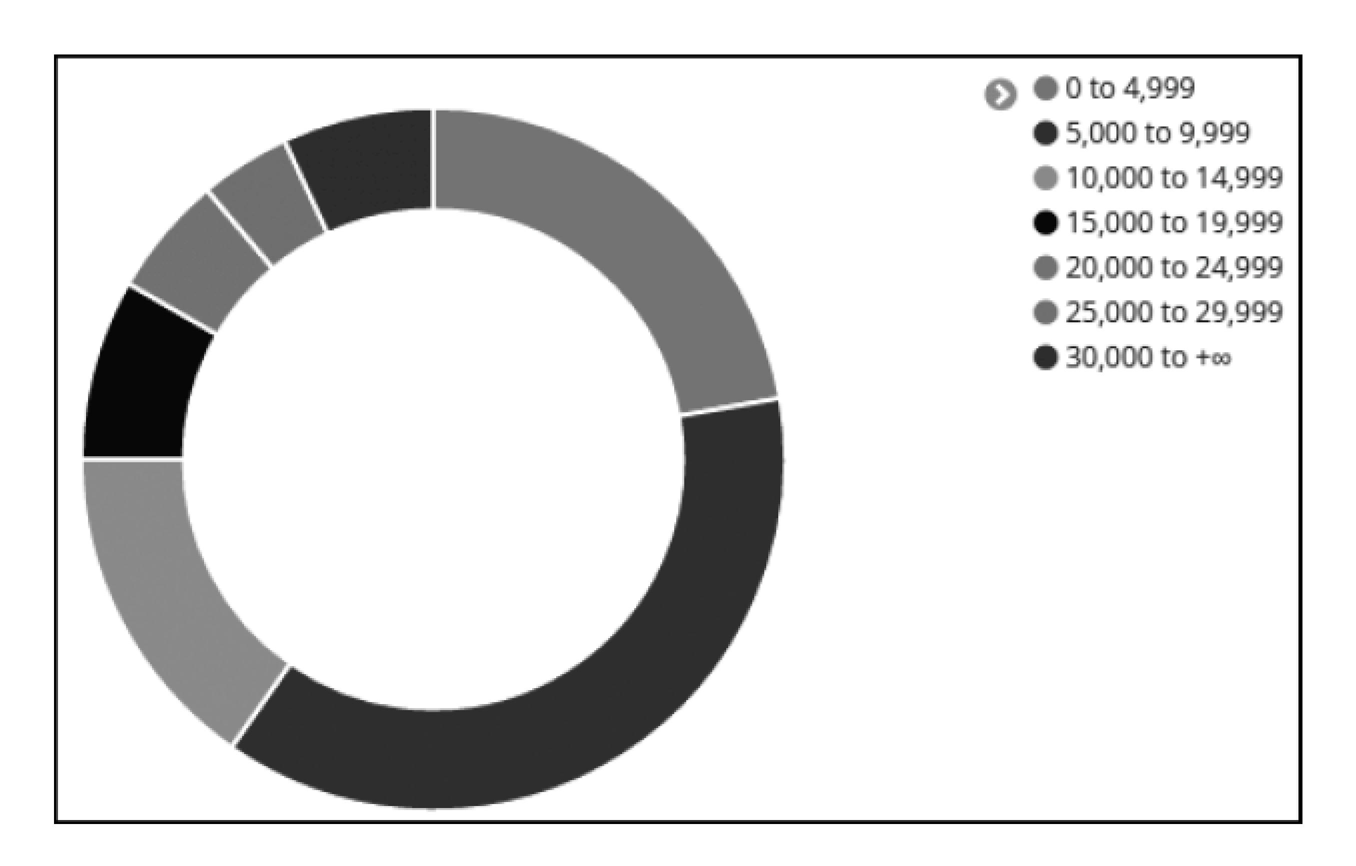


图 11.11　评测文章访问量饼图

以类似的方式也可以创建评测文章评论数的饼图。最后将 4 种统计图放入一个动态仪表板中并保存，如图 11.12 所示。

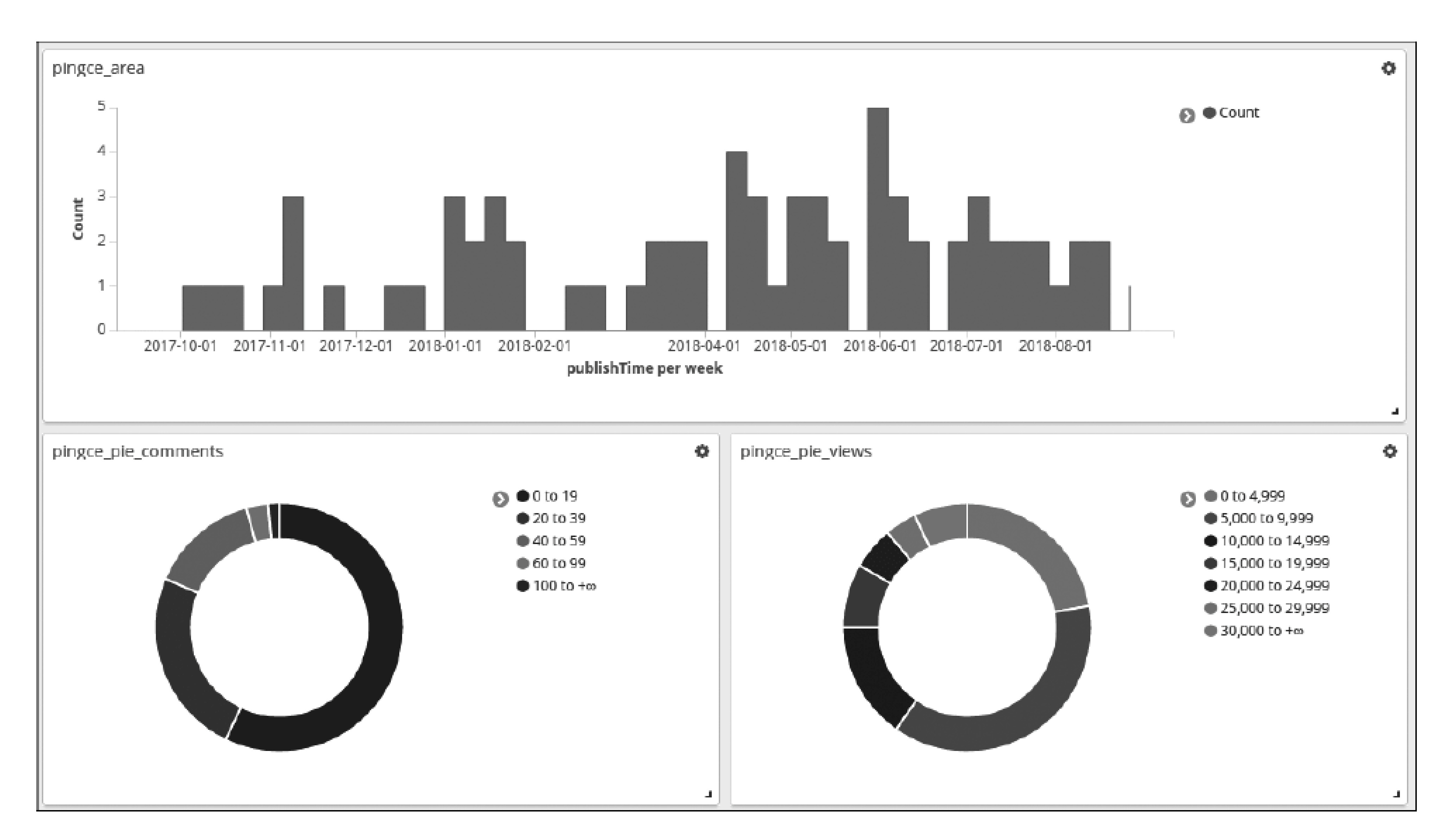


图 11.12　索引 pingce 的可视化展示

11.5.2　对 Logstash、Beats 的可视化展示

接下来对索引 logstash-*、filebeat-* 和 metricbeat-* 的文档数据进行统计。首先创

建一个折线统计图，在 Visualize 组件中选择 Line chart，设置横轴聚合方式为 Date Histogram，其他选项保留默认设置，单击▶按钮生成统计图，如图 11.13 所示。该图直观地反映了搜索引擎项目中访问量随时间的变化情况。

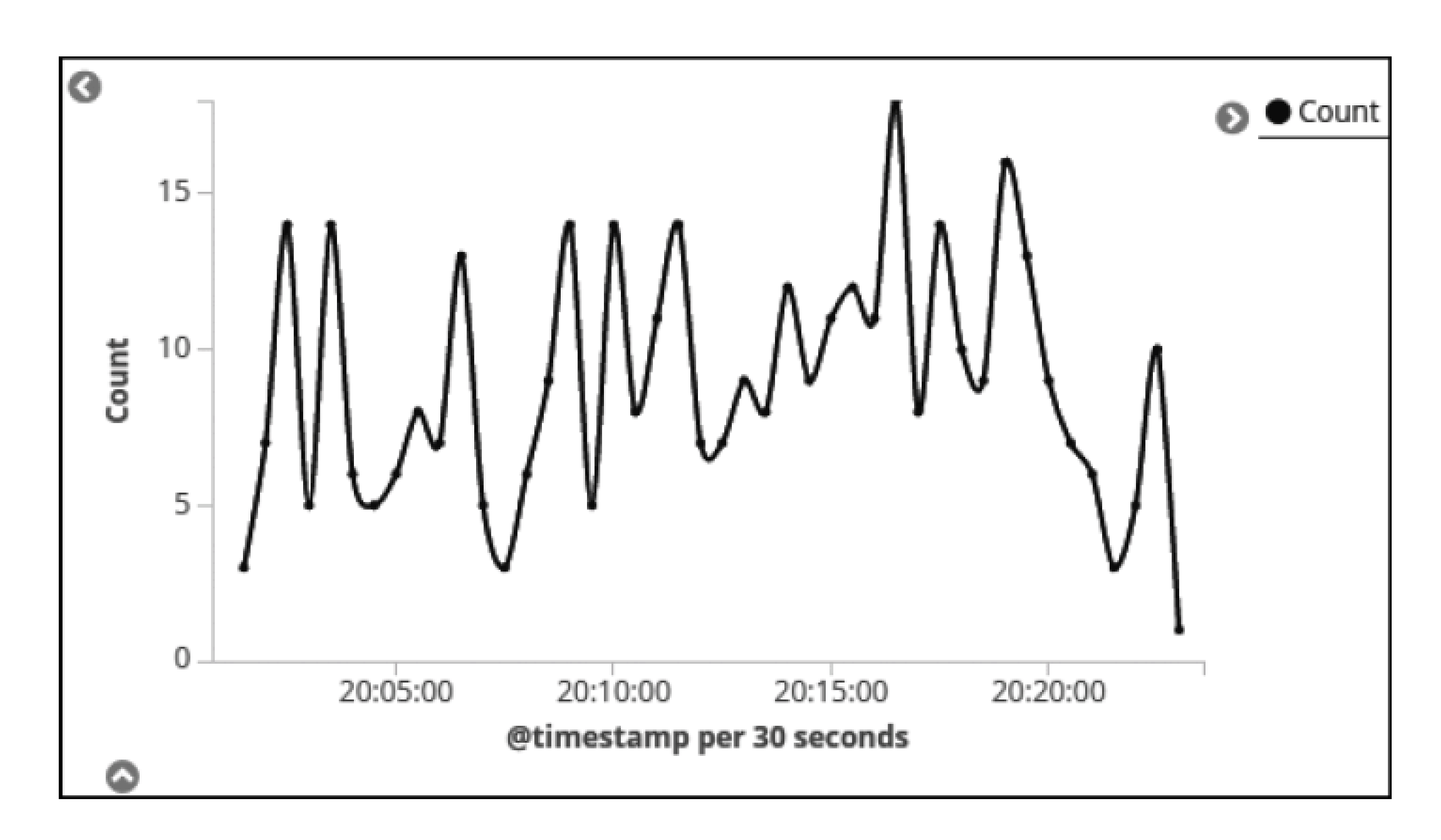


图 11.13　搜索引擎项目访问量随时间变化的折线图

接下来创建饼图，对 Filebeat 在不同日志文件中采集到的信息量进行统计。在饼图的 Buckets 设置中选择聚合方式 Terms，在 Field 中选择字段 Source，其他选项保留默认设置，生成后的饼图如图 11.14 所示。该图显示，Filebeat 采集了两种日志文件的数据，前缀为 catalina 的两个日志文件数据量较大，而前缀为 localhost 的两个日志文件数据量较小。

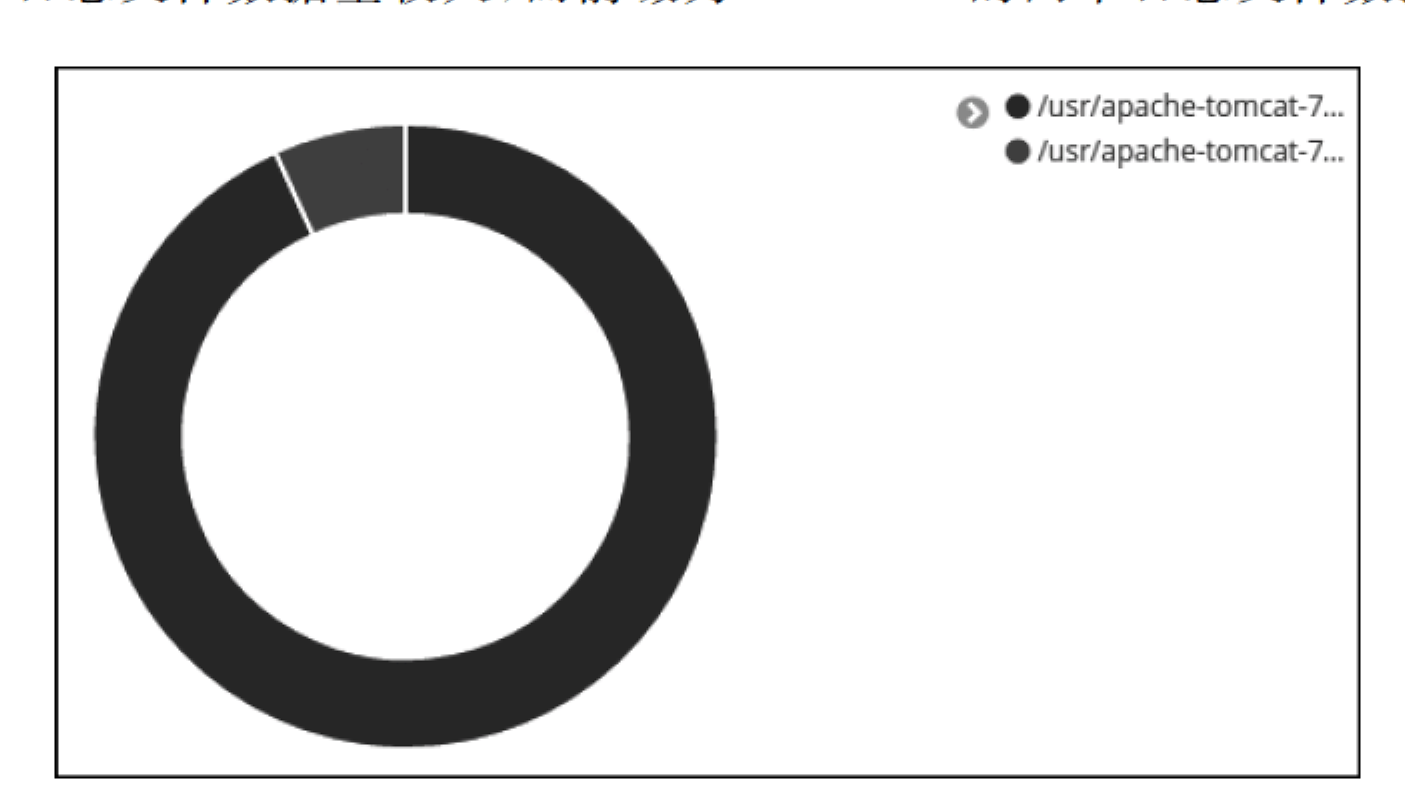


图 11.14　Filebeat 采集日志来源饼图

创建 Metricbeat 采集数据的时间线，在 Visualize 组件中创建一个 Timeseries，在 timelion expression 输入框中输入如下查询表达式：

```
.es(index='metricbeat-*',timefield='@timestamp')
```

设定时间间隔为分钟，运行该查询表达式，可以得到如图 11.15 所示的时间线。图中清晰显示了 Metricbeat 执行采集和传输任务最集中的时间段。

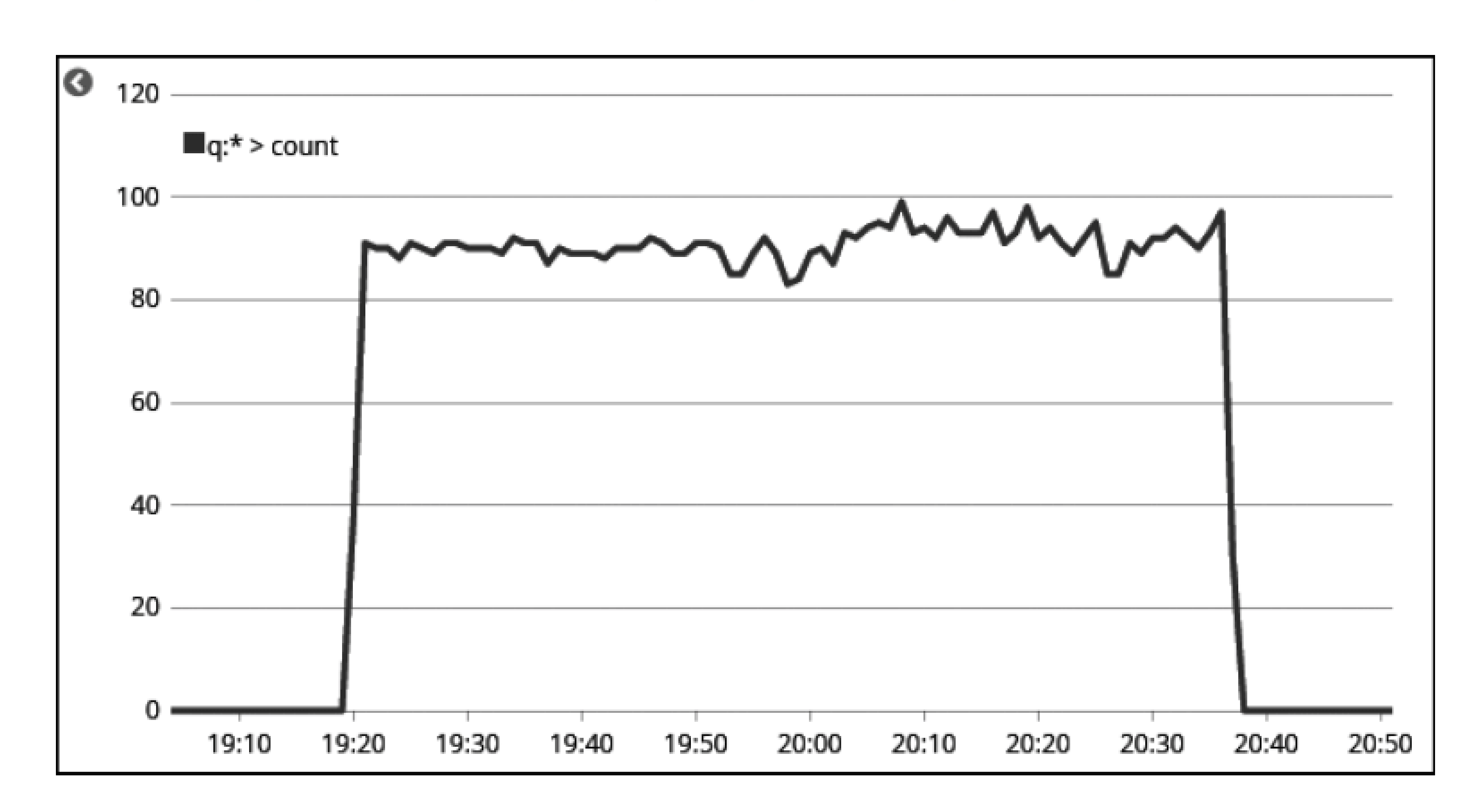


图 11.15　Metricbeat 采集数据的时间线

在上述统计图表之外，也可利用饼图来统计 Logstash、Filebeat 和 Metricbeat 在不同时间段采集的文档数量，创建方法与 11.5.1 节中饼图的创建方法类似，只需执行 Date Range 聚合将数值换成时间即可。针对各个饼图也可以创建统计数值以显示各自的文档总数。最后将创建的各种可视化统计图表放入动态仪表板，如图 11.16 所示。

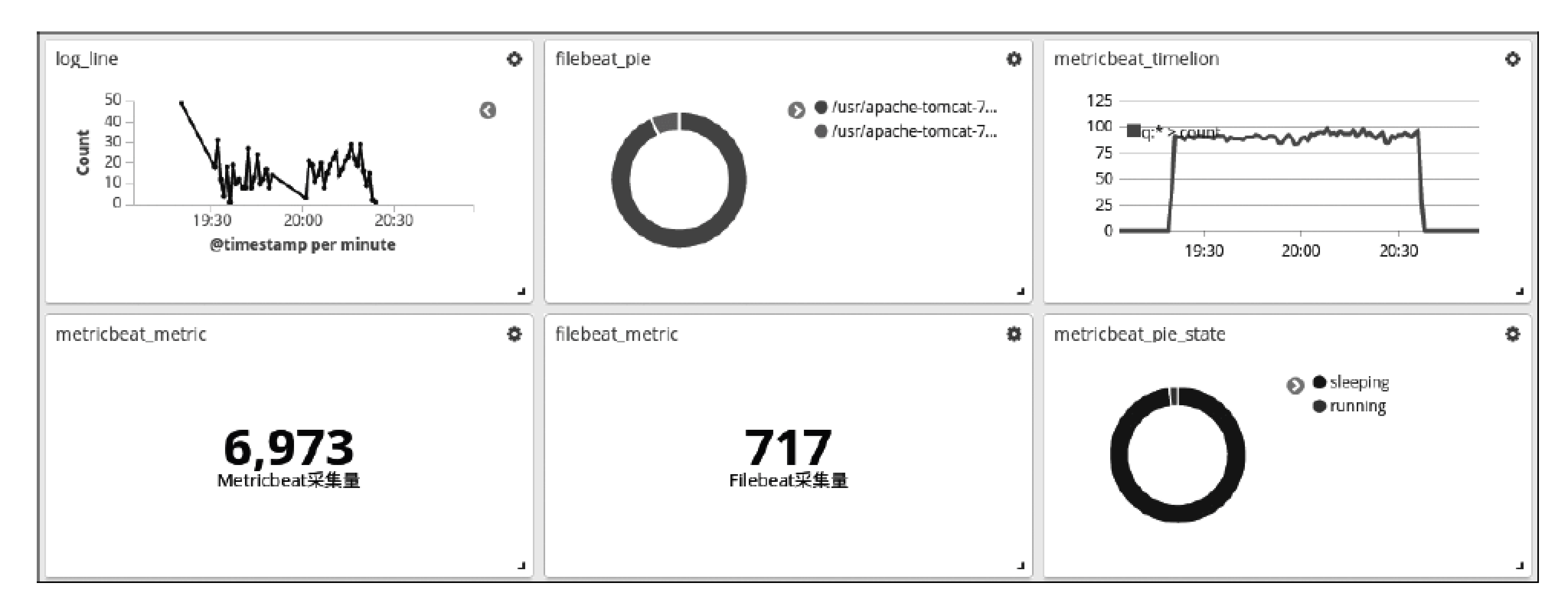


图 11.16　有关 Logstash 和 Beats 的可视化展示

11.6　基于 X-Pack 的系统监控

登录 Kibana，单击界面左侧导航栏中的 Monitoring 导航按钮，即可查看 Elasticsearch 和 Kibana 当前的总体运行状态数据，如图 11.17 所示。

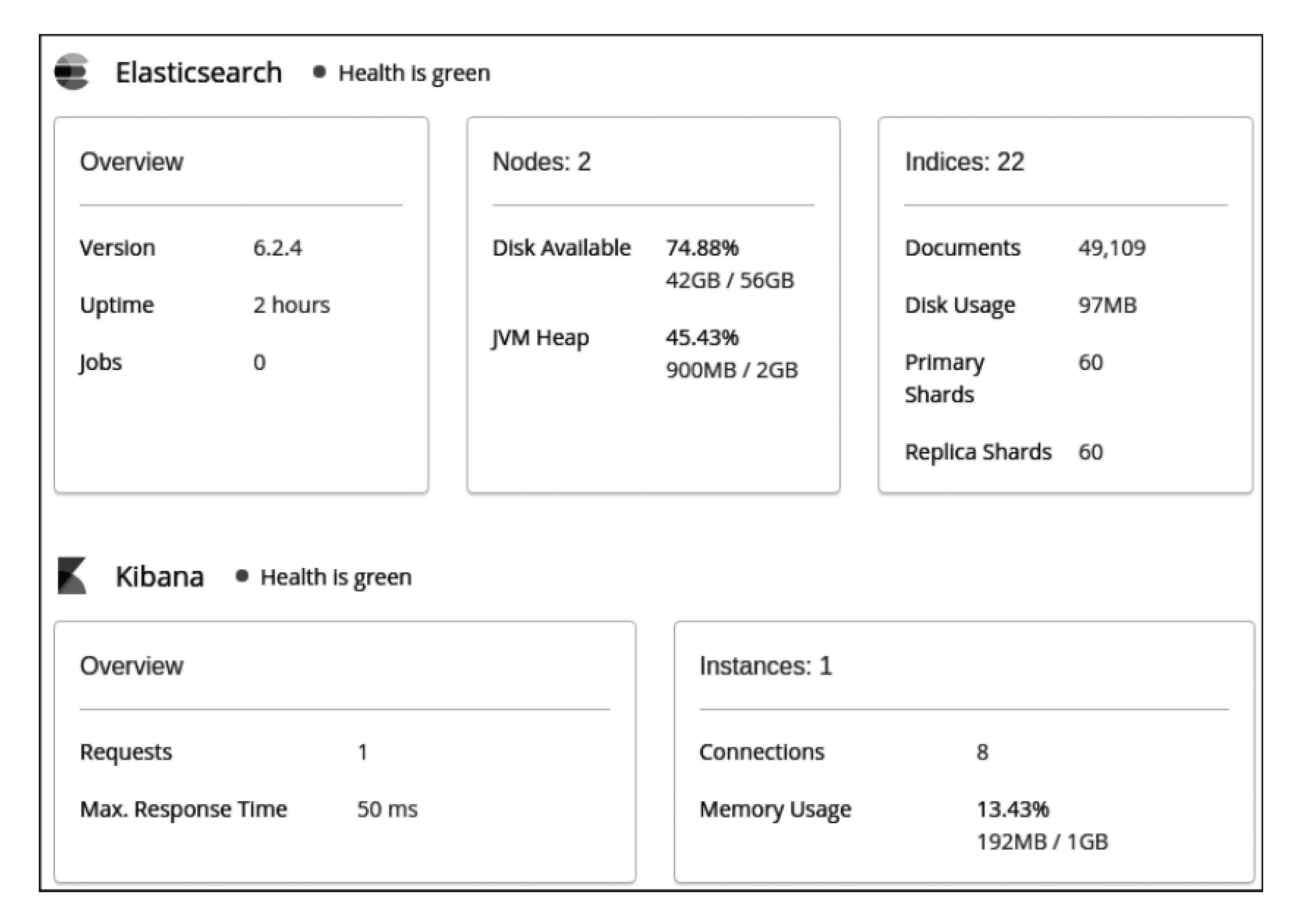


图 11.17　系统总体运行状态数据

在 Elasticsearch 的索引文件列表中，可以查看索引文件 pingce 的运行状态，如图 11.18 所示。图中的 Search Rate 折线图反映了项目中执行搜索的情况，其余图中的线保持稳定值不变。

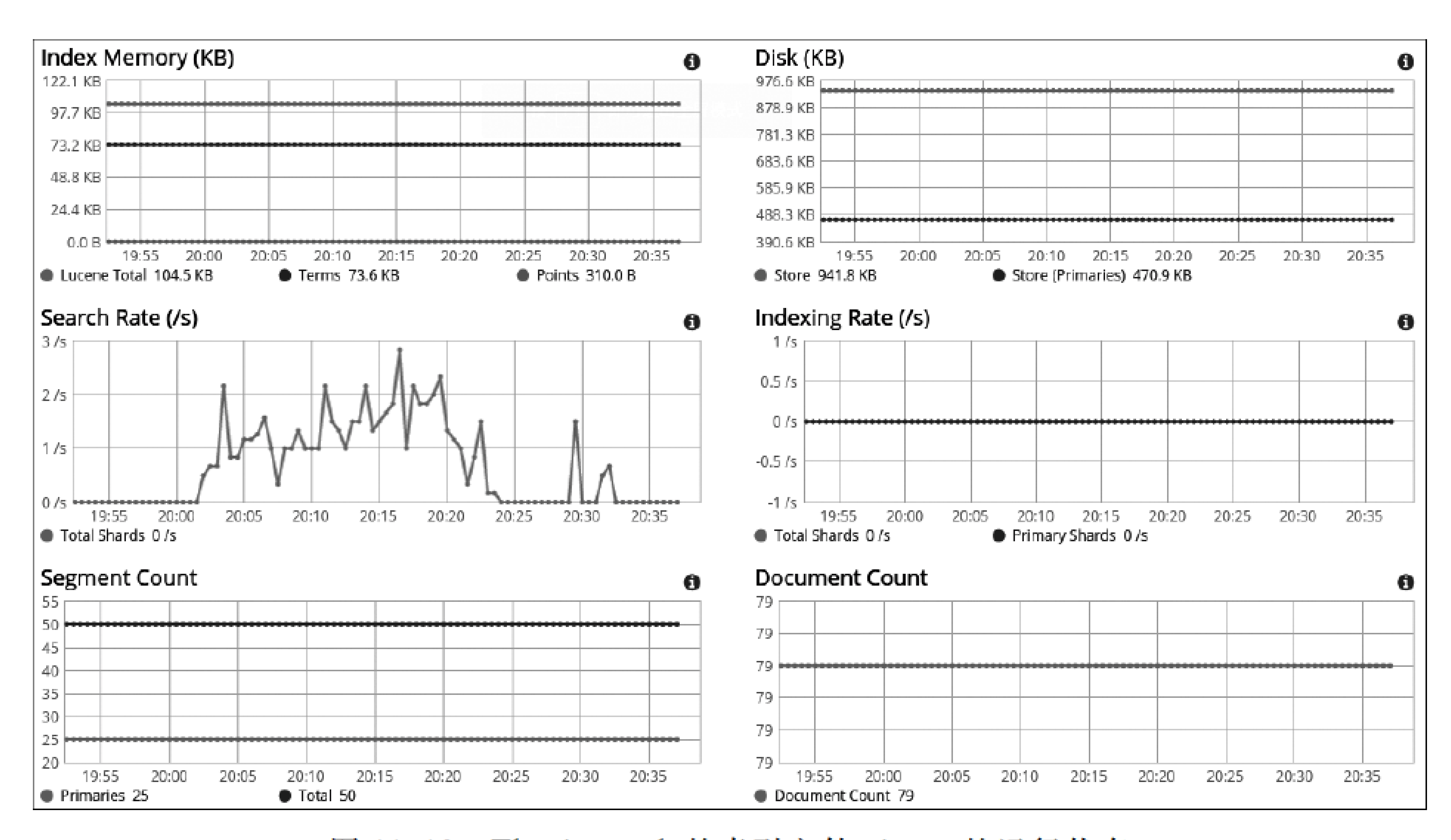


图 11.18　Elasticsearch 的索引文件 pingce 的运行状态

折线图下方展示了 Elasticsearch 节点中不同分片的运行情况，如图 11.19 所示。从图中不难看出，系统中开启了两个节点，每个节点中 5 个分片均工作正常。

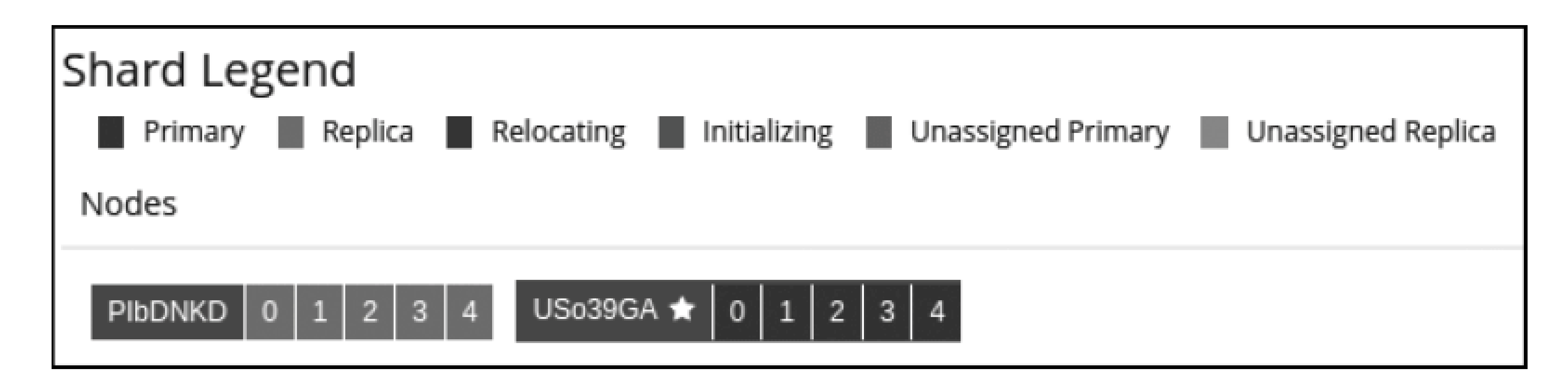


图 11.19　节点和分片运行状态

在 Indices 列表中单击 filebeat-* 索引可以查看 Filebeat 组件的运行状态，如图 11.20 所示。其中的 Search Rate 图线的数值基本为零，表示之前基本没有对 Filebeat 的查询。从其他折线图的走势中均可看出由于 Filebeat 对日志数据的处理而产生的性能波动，从 Document Count 中可以看出采集处理的信息正处于一个不断增长的趋势，这从侧面反映了 Elasticsearch 正不断处理来自客户端的搜索请求。

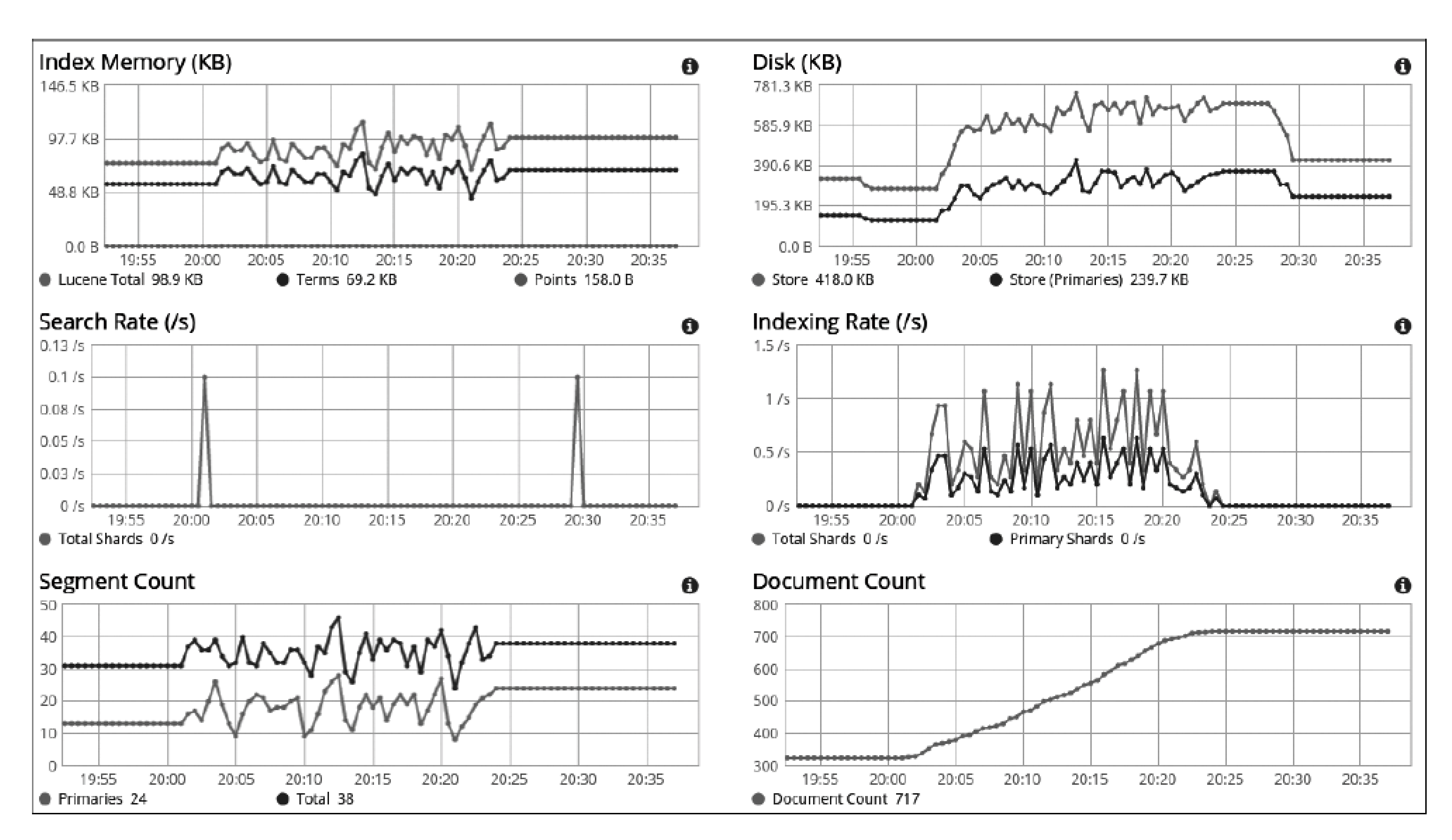


图 11.20　Filebeat 运行状态

在 Monitoring 程序界面中单击 Kibana 的 Instances 链接，可查看 Kibana 实例的运行状态，如图 11.21 所示。其中的 System Load 折线图反映了系统负载基本处于较为平稳的状态，从其余折线图中也可以看出系统响应和连接相关指标走势情况。

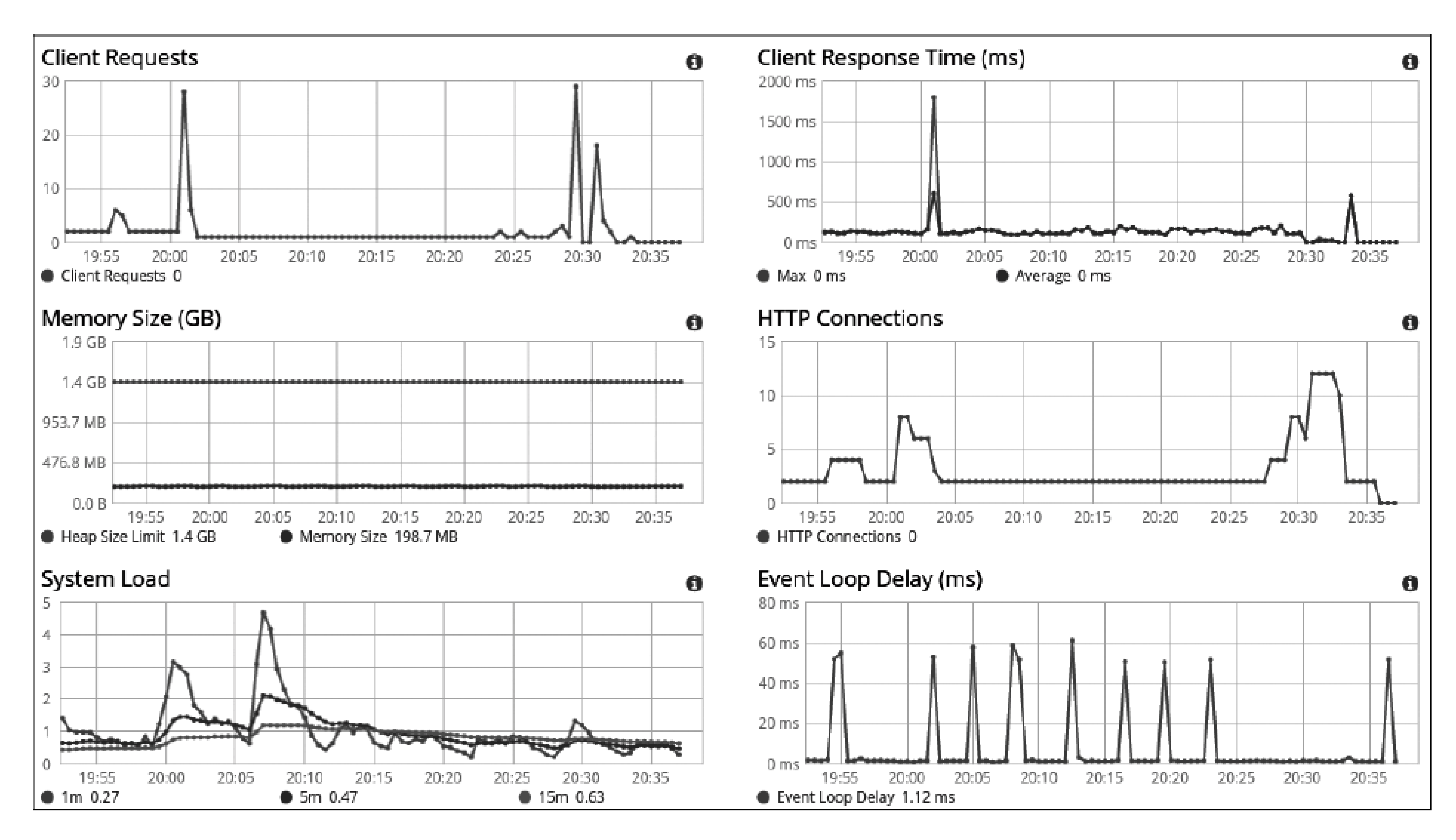


图 11.21　Kibana 实例运行状态

11.7　扩展知识与阅读

Selenium 是一个用于 Web 应用程序测试的工具。Selenium 测试直接运行在浏览器中。有关 Selenium 的详细内容可参阅官网 http://www.seleniumhq.org/。另外，文献（曾宪杰，2014）给出有关大型网站系统与 Java 中间件的相关实践技术，而有关中间件的理论背景可参阅文献（Andrew，2007）。除此之外，软件负载中心与集中配置管理也是需要考虑的问题。

11.8　本章小结

本章给出一个基于 6.2 版 Elasticsearch、Logstash、Kibana、X-Pack 和 Beats 的网络信息检索、日志分析及可视化管理的工程实例。其中，网络信息采集是利用 Selenium 模拟完成的，随后将采集到的信息后存入 Elasticsearch 索引中；日志分析使用的是 Logstash，依照配置文件中的配置信息进行日志信息的输入、解析和输出；日志文件和系统指标数据利用 Beats 相关组件传输到 Elasticsearch 索引中；由 Kibana 完成各个索引中文档数据的可视化展示；对于 Elasticsearch 索引运行状态、Logstash 数据处理情况以及 Kibana 实例运行状况等的图形化监控，使用 X-Pack 中的 Monitoring 程序完成。通过对本章的学习，希望读者对 Elastic Stack 全套组件的开发和运作有进一步的了解，以便在今后的实战环境中熟练使用大数据搜索与分析相关工具解决实际项目中的问题。

参考文献

[1] Andlinger P,Gelbmann M. 2015. MongoDB is the DBMS of the year,defending the title from last year [EB/OL]. http://db-engines. com/en/blog_post/41.

[2] Andrew C. 2013. Exploring Elasticsearch: A Human-Friendly Approach to Elasticsearch[Z].

[3] Andrew S T,Maarten V S. 2007. 分布式系统原理与范型[M]. 北京:清华大学出版社.

[4] BigdataTina2016. 2016. 开源搜索引擎 Elasticsearch 5.0 版本正式发布[EB/OL]. http://mt. sohu. com/20161104/n472267528. shtml.

[5] geloin. 2015. 分布式搜索 Elasticsearch——搜索(二)[EB/OL]. http://blog. csdn. net/geloin/article/details/8908238.

[6] Gormley C, Tong Z. 2015. Elasticsearch 权威指南[EB/OL]. Gavin Foo, 译. http://fuxiaopang. gitbooks. io/learnelasticsearch/.

[7] laigood. 2013a. Elasticsearch 插件大全[EB/OL]. http://www. searchtech. pro/elasticsearch-plugins.

[8] laigood. 2013b. 一些国外优秀的 Elasticsearch 使用案例[EB/OL]. http://www. searchtech. pro/elasticsearch-users-case.

[9] Michael M,Erik H,Otis G. 2011. Lucene 实战[M]. 2 版. 牛长流,等译. 北京:人民邮电出版社.

[10] Rafal K,Marek R. 2015. ElasticSearch:可扩展的开源弹性搜索解决方案[M]. 北京:电子工业出版社.

[11] Ricardo BY,Berthier RN. 2012. 现代信息检索[M]. 黄萱菁,等译. 北京:机械工业出版社.

[12] Spring. 2018. Spring Boot[EB/OL]. https://spring. io/projects/spring-boot.

[13] Steven JM,William C W. 2013. Java 设计模式[M]. 张逸,等译. 北京:电子工业出版社.

[14] Turnbull J. 2015. The Logstash Book: Log Management Made Easy[EB/OL]. http://cds. cern. ch/record/2149837/files/?ln=zh_CN.

[15] 百度百科. 2014a. Elasticsearch[EB/OL]. http://baike. baidu. com/view/8005387. htm?fr=aladdin.

[16] 百度百科. 2014b. Maven[EB/OL]. http://baike. baidu. com/view/336103. htm?fr=aladdin.

[17] 百度百科. 2018. PIP(Python 包管理工具)[EB/OL]. https://baike. baidu. com/item/PIP/20435212?fr=aladdin.

[18] 高凯,仇晶,张晓明,等. 2014. 信息检索与智能处理[M]. 北京:国防工业出版社.

[19] 韩陆. 2014. Java RESTful Web Service 实战[M]. 北京:机械工业出版社.

[20] 黄健宏. 2014. Redis 设计与实现[M]. 北京:机械工业出版社.

[21] 开源中国社区. 2014. 项目构建工具 Maven[EB/OL]. http://www. oschina. net/p/maven.

[22] 李刚. 2014. 疯狂 Ajax 讲义[M]. 北京:电子工业出版社.

[23] 李志义. 2015. 网站信息组织优化:基于网络日志的用户行为分析[M]. 北京:电子工业出版社.

[24] 李子骅. 2013. Redis 入门指南[M]. 北京:人民邮电出版社.

[25] 廖振,黄亚楼,刘杰. 2015. 精彩纷呈的网络搜索日志挖掘[J]. 中国计算机学会通讯,10(5),25-30.

[26] 罗刚. 2014. 解密搜索引擎技术实战[M]. 北京：电子工业出版社.
[27] 饶琛琳. 2015a. ELK Stack 权威指南[M]. 北京：机械工业出版社.
[28] 饶琛琳. 2015b. Kibana 中文指南[EB/OL]. http://kibana.logstash.es/.
[29] 王继民. 2014. Web 用户查询日志挖掘与应用[M]. 北京：知识产权出版社.
[30] 王练. 2017. Elasticsearch 6.0.0 正式发布，带来大量新特性[EB/OL]. https://www.oschina.net/news/90590/elasticsearch-6-0-0-released.
[31] 微软. 2017. AboutExecutionPolicies[EB/OL]. https://technet.microsoft.com/zh-CN/library/hh847748.aspx.
[32] 吴军. 2013. 数学之美[M]. 北京：人民邮电出版社.
[33] 许晓斌. 2011. Maven 实战[M]. 北京：机械工业出版社.
[34] 杨传辉. 2014. 大规模分布式存储系统原理解析与架构实战[M]. 北京：机械工业出版社.
[35] 余晟. 2012. 正则指引[M]. 北京：电子工业出版社.
[36] 运维生存时间. 2015. ELK beats 平台介绍[EB/OL]. http://www.ttlsa.com/elk/elk-beats-platform/.
[37] 曾宪杰. 2014. 大型网站系统与 Java 中间件实践[M]. 北京：电子工业出版社.
[38] 张华平，高凯，黄河燕，等. 2014. 大数据搜索与挖掘[M]. 北京：科学出版社.
[39] 周宁，张玉峰，张李义. 2005. 信息可视化与知识检索[M]. 北京：科学出版社.
[40] 周苏，王文. 2016. 大数据可视化[M]. 北京：清华大学出版社.